IMPLEMENTASI *SHARED SESSION* DALAM KLASTER *SERVER WEB* MENGGUNAKAN *PHP* DAN *MYSQL*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
R. Moch Makruf Puja Pradana
NIM: 145150207111060

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2019

PENGESAHAN

Implementasi *Shared Session* Dalam Klaster *Server Web* Menggunakan *PHP* dan *MySQL*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh:
R. Moch Makruf Puja Pradana
NIM: 145150207111060

Skripsi ini telah diuji dan dinyatakan lulus pada
10 Mei 2019

Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Mahendra Data, S.Kom., M.Kom	Dany Primanita Kartikasari, S.T, M.Kom.
NIK: 201503 861117 1 001	NIP: 19771116 200501 2 003

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 26 Maret 2019

R. Moch Makruf Puja Pradana

NIM: 145150207111060

KATA PENGANTAR

Puji dan syukur penulis panjatkan atas kehadirat Allah SWT karena rahmat dan karunia-Nya, penulis dapat menyelesaikan penelitian yang digunakan sebagai syarat untuk memperoleh gelar Sarjana Komputer pada Program Studi Teknik Informatika pada Fakultas Ilmu Komputer Universitas Brawijaya. Penelitian yang berjudul "IMPLEMENTASI *SHARED SESSION* DALAM KLASTER *SERVER WEB* MENGGUNAKAN *PHP* DAN *MYSQL*" dengan tujuan mengetahui penerapan arsitektur shared session yang diimplementasikan pada beberapa klaster *server web* menggunakan *php* dan *mysql*.

Untuk kesempatan ini, penulis juga menyampaikan banyak terimakasih kepada pihak-pihak yang telah membantu penulis selama mengerjakan skripsi ini, diantaranya:

1. Bapak Mahendra Data, S.Kom., M.Kom. dan Ibu Dany Primanita Kartikasari, S.T, M.Kom. selaku dosen pembimbing skripsi atas segala waktu, tenaga, dan ilmu yang telah diberikan untuk membimbing dan mengarahkan penulis dalam proses penyusunan skripsi ini.
2. Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D., Bapak Ir. Heru Nurwasito M.Kom, Bapak Suprapto, S.T, M.T, dan Bapak Edy Santoso, S.Si, M.Kom selaku Dekan, Wakil Dekan I, Wakil Dekan II, dan Wakil Dekan III Fakultas Ilmu Komputer Universitas Brawijaya.
3. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D, Bapak Agus Wahyu Widodo, S.T, M.Cs, dan Bapak Muhammad Tanzil Furqon, S.Kom, M.CompSc selaku Ketua Jurusan, Ketua Program Studi dan Sekretaris Program Studi Teknik Informatika.
4. Seluruh dosen dan staff Fakultas Ilmu Komputer Universitas Brawijaya atas segala waktu, tenaga, dan ilmu yang diberikan kepada penulis.
5. Kepada kedua orang tua dan seluruh keluarga besar atas segala nasehat, kasih sayang, perhatian, dan semua pengorbanan yang telah diberikan kepada penulis.
6. Teman-teman Teknik Informatika yang membantu selama proses perkuliahan di Fakultas Ilmu Komputer Universitas Brawijaya.
7. Fariz F dan Rexa Mei B yang meluangkan waktunya untuk membantu menyelesaikan permasalahan sistem yang terdapat dalam skripsi.

8. Teman – teman kontrakan selaku teman terdekat penulis selama di Malang yang bersama-sama menempuh pendidikan dan memberikan semangat moral kepada penulis di dalam maupun luar lingkungan Fakultas Ilmu Komputer Universitas Brawijaya Malang.

Penulis menyadari bahwa dalam penyusunan laporan skripsi ini masih banyak kekurangan dan jauh dari sempurna. Oleh karena itu, kritik serta saran yang membangun sangat penulis harapkan. Akhir kata, penulis berharap laporan skripsi ini dapat memberikan manfaat bagi semua pihak.

Malang, 26 Maret 2019

Penulis

jendralmakruf@gmail.com

ABSTRAK

Session, merupakan cara yang digunakan untuk menyimpan informasi pada komputer *server* untuk digunakan pada beberapa halaman termasuk halaman itu sendiri. Dalam penggunaan klaster *web server* mampu menghasilkan kinerja yang lebih baik daripada menggunakan *server* tunggal yang meng*handle* sebuah *website*. Namun terdapat masalah yang timbul di dalam pengembangan antara klaster *web server* terhadap penggunaan *session* itu sendiri. Pada klaster *web server* yang diwakilkan oleh satu *node server* akan berjalan secara independen, jika *session* awal yang dibentuk dalam aplikasi *web* tersebut hanya didalam satu *node server* saja maka *node server* lainnya tidak bisa mendapatkan akses *session* yang sama. Solusi yang bisa digunakan ialah menyimpan data *session* tersebut kedalam sebuah *database* berupa *MySQL* serta akses data *session* dapat dibagikan ke *web server* lainnya. Terungkap permasalahan lagi bagaimana satu klaster *web server* dapat berkomunikasi terhadap database dalam menyimpan dan meminta data *session*. Dengan permasalahan tersebut, penelitian ini mengembangkan metode *Shared Session* yang akan diterapkan melalui *PHP*. Untuk membuktikan metode tersebut dapat menjalankan sistem sesuai fungsinya dilakukan dua buah pengujian yaitu pengujian *login* serta *logout* dan pengujian *black box*. Pada pengujian *login* dan *logout* yang direncanakan mampu menyelesaikan fungsi utamanya dalam menyimpan dan membagikan data *session* walaupun terdapat kondisi satu *server web* yang nonaktif. Pada pengujian *black box* didapatkan hasil *response time* di masing-masing skenario, pada skenario pertama dari 100 *request session* menghasilkan *response time* sebesar 18,7 *second* sampai dengan 26,2 *second*, pada skenario kedua dari 200 *request session* menghasilkan *response time* sebesar 1 *minute* sampai dengan 1,38 *minute*, pada skenario ketiga dari 300 *request session* menghasilkan *response time* sebesar 10,07 second sampai dengan 15,25 *second*.

Kata kunci: *session, shared session, PHP, MySQL*, *response time*.

ABSTRACT

Session is a method that used to store information on a computer server that can be used on several pages including the page itself. In using a web server cluster, it can produce a better performance than using a single server that handles a website. But there are some problems that arise in the development between the web server cluster against the use of the session itself. In a web server cluster is a represented by a single node will run independently, if the initial session formed in the web application only have one node server, then the other server nodes can't get the same session access. The solution that can be used is to save the data session into database with mySQL, and the access of data session can be shared to another web servers. Another problem was revealed is how one web server cluster can communicate with the database in storing dan requesting data sessions. With these problems, this study developed the Shared Session method that will be applied through PHP. To prove that the method can run the system according to function, two tests are applied, login and logout testing and black box testing. In testing the login and logout that is designed can be fulfilled in storing and sharing data sessions even though there is a condition of one disabled web server. In the black box testing the response time results obtained in each scenario, in the first scenario of 100 session requests produces a response time of 18.7 second to 26.2 second, in the second scenario of 200 session requests produces a response time of 1 minutes up to 1.38 minutes, in the third scenario of 300 session requests to produce a response time of 10.07 seconds to 15.25 seconds.

Keyword: session, shared session, PHP, MySQL, response time.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Session adalah cara yang digunakan untuk menyimpan informasi pada komputer *server* untuk digunakan pada beberapa halaman termasuk halaman itu sendiri. *Session* menyimpan informasi ke dalam bentuk variabel *super global $_SESSION*. Variabel ini disimpan pada komputer *server* dan dapat digunakan oleh semua halaman pada *website* tempat *session* dimulai. *Session* berbeda dengan *cookies* yang menyimpan informasi pada komputer *client* (pengguna) (Mukhlis, 2018). Session merupakan konsep abstrak yang merepresentasikan interaksi antara *browser* dan *server*. Fungsi *session* tersebut yakni untuk menjaga atau memelihara informasi akses dari seorang pengakses/pemakai aplikasi *web*.

Session ini umumnya dibangun atas *cookies*, yang dimana *session* terbentuk berdasarkan dari kombinasi antara *client-side session ID* dan *server-side session* data. Kemudian dari kedua variabel tersebut *client-side session ID* yang paling umum disimpan dalam *cookies* sedangkan *server-side session* data disimpan dalam *file* data *log* pada *server* berupa *temporary file*. *Cookie* awal diberikan pada saat *web browser* pertama kali dibuka dan melakukan *request* halaman *web* ke *server*, setelah *server* dapat mengenali *IP client* kemudian *server* akan merespon dengan memberikan data *session* dalam bentuk *session ID.* Selanjutnya, setelah itu *client* akan mengirim kembali *session ID* ke *server* untuk mendapatkan data *client* yang sudah tersimpan pada *database server*.

Disaat pengguna *web server* semakin meningkat dengan seiring berjalannya waktu maka dibutuhkan *server* yang dapat berkerja optimal. Jika *web server* yang dimiliki hanya ada satu, maka memungkinkan terjadinya "*a single point of failure*" (SPOF) yaitu kondisi *server* yang gagal merespon maka sistem akan tidak berfungsi. Hal itu bisa terjadi karena terlalu banyak *request* yang harus di *handle* oleh satu buah *web server* tadi (Julianto, 2017). Maka dibuatlah sebuah arsitektur jaringan dalam *web server* yang disebut arsitektur *multiple server* yaitu kumpulan dari beberapa *web server* yang saling terhubung dan bekerja sama, yang memiliki fungsi untuk mencapai keandalan (*reliability*) dan ketersediaan (*availability*) yang tinggi (Singh, 2015).

Dalam mengembangkan dan memaksimalkan penggunaan klaster *web server* yang dapat menghasilkan kinerja lebih baik daripada menggunakan *server web* tunggal. Klaster *web server* merupakan metodologi pengelompokan *server* yang independen untuk memberikan fleksibilitas, skalabilitas dan ketersediaan yang lebih baik. Serta

mampu untuk menangani dan memperoleh kebutuhan akses yang cepat dan handal untuk *client* utamanya jika berkaitan dengan *session.* Namun terdapat masalah yang timbul di dalam pengembangan antara klaster *web server* terhadap penggunaan *session* itu sendiri. Dalam setiap klaster *web server* yang direpresentasikan melalui satu *node server* akan berjalan secara independen, jika *session* awal yang dibentuk dalam aplikasi *web* tersebut hanya berada dalam satu *node server* saja, maka *node server* lainnya tidak bisa mendapatkan akses *session* yang sama. Maka diperlukan *node server* alternatif untuk membantu kinerja dalam klaster *web server* yakni *server database*, *node server* tersebut berfungsi untuk menyimpan *session* data dan mampu dibagikan ke *node server* yang lainnya.

Dari dampak penggunaan klaster *server web* tersebut terhadap penyimpanan *session* belum sepenuhnya terselesaikan jika hanya memaksimalkan penggunaan *server database*. Peneliti ingin menambahkan sebuah metode yakni *Shared Session* yang bertujuan untuk membagikan data *session* yang telah tersimpan di *database* melalui *node server* sebelumnya agar *node server* lainnya mendapatkan akses *session* yang sama. Selain itu terdapat solusi metode yang lain yakni menggunakan metode *MemCached* dikarenakan metode ini juga mengadopsi penggunaan *redis cloud* (Lahtinen, 2014). Namun karena keterbatasan perangkat yang digunakan, maka peneliti dianjurkan menggunakan metode *shared session* beserta *server database* berupa *MySQL*.

Penelitian ini bertujuan ingin memberikan solusi dalam mengatasi permasalahan tersebut. Yakni dengan manajemen informasi *session* yang telah dibangun antara entitas *client* dan *server* dan disimpan ke dalam *database* berupa *MYSQL*, sekaligus membagikan kepada *node server* lain saat membutuhkan *session* tersebut dan *load balancing* berperan untuk membagikan *request* berupa data *session* dalam waktu bersamaan saat dibutuhkan oleh *client* kemudian dibuat *session* pada masing-masing *node server*. Metode *shared session* beserta *server database* ini dapat mencegah jika salah satu *node server* yang diarahkan oleh *load balancing* ini mati atau tidak aktif kemudian dapat di *handle* oleh *node server* yang ada ataupun masih aktif, karena *session* data masih disimpan dalam *database*. Sehingga pada tugas akhir ini penulis menawarkan penilitian tentang “IMPLEMENTASI SHARED SESSION DALAM KLASTER SERVER WEB MENGGUNAKAN PHP DAN MYSQL”.

1.2 Rumusan masalah

Berdasarkan latar belakang yang telah dikemukakan maka permasalahan yang akan dibahas dalam penelitian ini adalah :

1. Bagaimana mengimplementasikan *shared session* pada klaster *web server* menggunakan *PHP* dan *MySQL*?
2. Bagaimana kinerja dari hasil implementasi *shared session* menggunakan *PHP* dan *MySQL* pada klaster *web server*?

1.3 Tujuan

Adapun tujuan dari mengimplementasikan sistem ini secara umum, adalah :

1. Untuk mengatasi permasalahan *shared session* pada klaster *web server* menggunakan *PHP* dan *MySQL*.
2. Untuk mengetahui kinerja dari *shared session* pada klaster *web server* menggunakan *PHP* dan *MySQL*.

1.4 Manfaat

Penulisan skripsi ini diharapkan mempunyai manfaat yang baik dan berguna bagi pembaca dan penulis. Adapun manfaat yang diharapkan adalah sebagai berikut:

1. Dapat digunakan untuk mengatasi permasalahan *shared session* pada klaster *web server*.
2. Dapat mengetahui kinerja dari *shared session* pada klaster *web server* yang dibangun menggunakan *PHP* dan *MySQL*.

1.5 Batasan masalah

Adapun beberapa batasan masalah dalam penulisan skripsi ini yakni sebagai berikut:

1. Algoritme *load balancing* yang digunakan adalah berbasis *Round-Robin*.
2. Sistem *shared session* yang akan diimplementasikan pada penelitian ini, diterapkan melalui aplikasi *website* yang menggunakan bahasa pemrograman berbasis *PHP*.
3. Sistem *shared session* hanya diterapkan pada 4 *web server* aktif dalam satu klaster *server web*.
4. *Database* berbasis *MySQL* digunakan sebagai pusat penyimpanan seluruh data *user* termasuk utamanya data *session*.

1.6 Sistematika pembahasan

Sistematika pembahasan merupakan penjabaran deskriptif mengenai hal hal yang akan dibahas dalam penelitian ini, penelitian ini disusun sebagai berikut:

BAB I PENDAHULUAN

Dalam bab ini penulis menguraikan latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika pembahasan.

BAB II LANDASAN KEPUSTAKAAN

Dalam bab ini penulis membahas mengenai kajian pustaka dan teori-teori yang diperlukan didalam penelitian.

BAB III METODOLOGI PENELITIAN

Dalam bab ini penulis membahas metode atau langkah-langkah yang dilakukan dalam penelitian yang akan dilaksanakan.

BAB IV PERANCANGAN

Dalam bab ini penulis membahas rancangan program yang disajikan dalam bentuk diagram alir dan juga rancangan struktur data yang akan digunakan dalam program.

BAB V IMPLEMENTASI

Dalam bab ini penulis membahas implementasi dari sistem yang sudah diterapkan.

BAB VI PENGUJIAN SISTEM

Bab ini akan menjelaskan skenario pengujian, hasil pengujian beserta analisa dari pengujian yang dilakukan.

BAB VII PENUTUP

Bab ini akan berisi kesimpulan yang merupakan rangkuman proses dan hasil dari keseluruhan penelitian dan saran yang berguna untuk pengembangan dan perbaikan untuk sistem yang ada.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini membahas tentang kajian pustaka dan dasar teori yang digunakan untuk menunjang dari penulisan Implementasi *Shared Session* Dalam Klaster *Server Web* Menggunakan *PHP* dan *MySQL.*

2.1 Kajian Pustaka

Pada sub bab ini dilakukan kajian pustaka terhadap penelitian sebelumnya yang terkait dengan penelitian ini dan terdiri dari beberapa referensi ilimiah penelitian yang terkait pada penelitian sebelumnya yang digunakan untuk penelitian ini. Referensi penelitian sebelumnya yang digunakan dijadikan sebagai pedoman dalam pelaksanaan penelitian. Berikut perbandingan penelitian terdahulu dan sekarang dapat dilihat pada tabel 2.1.

Tabel 2.1 Penilitian Ilmiah yang terkait

No	Nama Penulis, Tahun dan Judul	Persamaan	Perbedaan	
			Penelitian Terdahulu	Rencana Penelitian
1	Khairul Ansharullah. [2016]. IMPLEMENTASI SISTEM LOAD BALANCING DENGAN ALGORITMA ROUND ROBIN UNTUK MENGATASI BEBAN SERVER DI SMK NEGERI 2 KUDUS	Mengimplementasi *Load Balancer* menggunkan algoritme *Round Robin*	Menggunakan *Load Balancer* menggunkan algoritme *Round Robin*	Mengimplementasi pada *Load Balancer Round Robin* pada klaster *server web*
2	Handoko Yoga Hartomo. [2015]. IMPLEMENTASI WEB SERVER LOAD BALANCING PADA MESIN VIRTUAL	Mengimplementasi *Load Balancer* pada mesin virtual	Mengimplementasi *Load Balancer* pada mesin virtual dalam *host* yang sama	Mengimplementasi *Load Balancer* pada mesin virtual dalam *host* yang berbeda
3	Maya Rosalia. [2016]. IMPLEMENTASI HIGH AVAILABILITY SERVER MENGGUNAKAN METODE LOAD BALANCING DAN FAILOVER PADA VIRTUAL WEB SERVER CLUSTER	Mengimplementasi Load Balancer pada virtual klaster server web	Mengimplementasi *Load Balancer* menggunakan *HAPROXY* dan *NGINX*	Mengimplementasi pada *Load Balancer NGINX* serta *PHP* dan *MySQL*

4	Akbar Dwi Prastyo. [2017]. IMPLEMENTASI MOODLE MENGGUNAKAN LOAD BALANCING, FAILOVER, DAN SHARED SESSION	Mengimplementasi *Load Balancing* dan *Shared Session*	Implementasi *Moodle* Menggunakan *Load Balancing*	Mengimplementasi pada *Shared Session* pada klaster *server web*

Penelitian yang pertama berjudul "Implementasi Sistem Load Balancing Dengan Algoritma Round Robin Untuk Mengatasi Beban Server Di SMK Negeri 2 Kudus" (Ansharullah, 2016). Khairul Ansharullah menawarkan penelitian mengenai pengembangan dalam bidang jaringan komputer telah dilakukan oleh SMK Negeri 2 Kudus. Akan tetapi masih banyak mengalami masalah salah satunya adalah beban server. Terlebih lagi di SMK Negeri 2 Kudus memiliki kurang lebih 800 orang pengguna. penelitian implementasikan sistem Load Balancing dengan tujuan untuk mengatasi beban server tersebut yang tidak sesuai dengan kapasitasnya dan untuk mengoptimalkan beban server sebelum dan sesudah penerapan sistem Load Balancing algoritma *Round Robin* pada *server* di SMK Negeri 2 Kudus. Dengan metode yang digunakan adalah metode Komparatif yaitu penelitian yang membandingkan dan menganalisa dua gejala atau lebih, membandingkan algoritma Least connection sebagai algoritma sebelumnya dengan algoritma *Round Robin*. Pengujian *Load Balancing* dengan dengan kedua algoritma menggunakan *software* yang bernama *Httperf. Httperf* menampilkan nilai sesuai parameter.

Penelitian yang kedua berjudul "Implementasi Web Server Load Balancing Pada Mesin Virtual" (Yoga Hartomo, 2015). Handoko Yoga Hartomo menawarkan penelitian mengenai penggabungan beberapa *server* (*cluster*) dengan teknik *Load Balancer*, bisa digunakan untuk mengatasi permasalahan tersebut karena teknik ini bekerja dengan membagi beban yang diterima oleh *server* dan ketika salah *server* mengalami kegagalan, maka anggota *cluster* yang aktif akan melayani permintaan dari *client*. Penelitian ini bertujuan untuk membuat *web server* dengan teknik *Load Balancer* pada mesin virtual. Aplikasi yang digunakan dalam penelitian ini adalah VMware, Ubuntu Server 14.10, Pound, HAProxy, dan Webserver Stress Tool 8.

Penelitian selanjutnya yang ketiga berjudul "Implementasi High Availability Server Menggunakan Metode Load Balancing Dan Failover Pada Virtual Web Server Cluster" (Rosalia, dkk., 2016). Maya Rosalia menawarkan penelitian mengenai implementasi *clustering* untuk *virtual web server* dan *high availability server* menggunakan metode *load balancing* dan *failover* untuk meningkatkan kehandalan dan ketersediaan layanan. Dalam implementasi nya, *load balancer* akan dikonfigurasi menggunakan *Haproxy* dan *Nginx* sebagai media *load balancing* untuk layanan HTTP. Kemudian akan dibandingkan performansi dua *software load balancing* dan *failover* dengan parameter yang akan diuji meliputi *throughput, request* per-detik, *request loss*, *cpu*

utilization dan *downtime*. Penelitian ini di latar belakangi karena semakin banyak yang mengakses melalui suatu situs web membuat beban kerja yang lebih pada suatu penyedia layanan yang disebut web server dan menjadi kurang optimal. Suatu single server bisa mengalami kegagalan yang disebabkan oleh meningkatnya jumlah *request* yang mencapai ribuan bahkan jutaan pada waktu yang bersamaan atau disebut dengan *overload*. Hal ini akan merugikan pihak yang mempercayakan situsnya pada suatu *web server*, karena situs tersebut tidak dapat diakses untuk waktu tertentu.

Penelitian yang keempat berjudul "Implementasi Moodle Menggunakan Load Balancing, Failover, Dan Shared Session" (Dwi Prastyo, 2017). Akbar Dwi Prastyo menawarkan penelitian mengenai implementasi dua *backend server* dengan menggunakan teknik *load balancing, failover,* serta *shared session*. Penelitian ini di latar belakangi bedasarkan oleh ujian yang merupakan kegiatan yang pasti dilakukan oleh setiap pelajar dan pengajar. Ujian yang sering dilakukan biasa nya bersifat tertulis atau manual, sangat jarang sekali dilakukannya ujian bersifat *online* atau otomatis. Tetapi ujian *online* untuk saat ini mulai digunakan tetapi masih menggunakan *single server*. Ujian menggunakan sistem *single server* masih banyak terdapat kekurangan, antara lain *respon time* lama, *server* banyak menampung beban, tidak ada *server* backup jika sewaktu-waktu *server* down. Dengan adanya *load balancing*, trafik dapat berjalan optimal, memaksimalkan *throughput*, memperkecil respontime dan menghindari *overload* pada salah satu jalur koneksi. Dengan diterapkannya *failover*, ketika salah satu *server down* atau mati maka *server* yang lain akan menggantikannya disaat proses *failover* terjadi fungsi *shared session* pun berjalan agar saat perpindahan *server* tidak perlu *login* ulang kembali.

2.2 Dasar Teori

Dalam sub bab dasar teori terdiri dari beberapa dasar teori sebagai penunjang pada saat melakukan penelitian. Dasar teori yang digunakan dapat dilihat sebagai berikut.

2.2.1 Web Server

Web server adalah sebuah perangkat lunak yang memberikan layanan berbasis data dan berfungsi menerima permintaan dari *HTTP* atau *HTTPS* pada *client* yang dikenal dan biasanya kita kenal dengan nama *web browser* lalu untuk mengirimkan kembali yang hasilnya dalam bentuk beberapa halaman *web* dan pada umumnya akan berbentuk dokumen *HTML*. Dalam bentuk sederhana *web server* akan mengirim data *HTML* kepada permintaan *web browser* sehingga akan terlihat seperti pada umumnya yaitu sebuah tampilan *website*.

Fungsi utama *web server* adalah untuk memproses atau melakukan transfer berkas permintaan pengguna melalui protokol komunikasi yang telah ditentukan sedemikian rupa. Halaman *web* yang diminta terdiri dari berkas teks, video, gambar,

file dan banyak lagi. Pemanfaatan *web server* berfungsi untuk mentransfer seluruh aspek pemberkasan dalam sebuah halaman *web* termasuk yang di dalam berupa teks, video, gambar atau banyak lagi.

Jika ada permintaan dari *browser*, maka *web server* akan memproses permintaan itu kemudian memberikan hasil prosesnya berupa data yang diinginkan kembali ke *browser*. Data ini mempunyai format yang standar, disebut dengan format *SGML (standar general markup language*). Data yang berupa format ini kemudian akan ditampilkan oleh *browser* sesuai dengan kemampuan *browser* tersebut (Lukitasari & Oklilas, 2013).

2.2.2 Layanan *Cluster*

Cluster service tidak bisa dilepaskan dari layanan *load balancing*, dan mempunyai tujuan untuk melakukan pencegahan kegagalan layanan bagi pengguna jaringan komputer bila salah satu sistem atau aplikasi yang ada dalam jaringan komputer mengalami kegagalan. Pada umumnya setelah layanan *load balancing* ini diimplementasikan maka *cluster service* juga diaplikasikan untuk membuat cadangan sistem atau aplikasi yang berjalan dalam jaringan komputer (Lukitasari & Oklilas, 2013).

Server Clustering merupakan salah satu solusi yang bisa diterapkan untuk mengatasi suatu permasalahan, yaitu teknologi yang menggabungkan beberapa *server* yang bekerja bersama-sama yang seolah-olah merupakan satu sistem tunggal. Terdapat beberapa metode pada sistem *clustering*, yaitu *load balancing* dan *failover*. Dengan *load balancing*, sistem akan dapat melayani beban pengaksesan yang besar dan meminimalisir kegagalan dalam melayani *request* dari *user* karena *load balancing* bekerja dengan mendistribusikan secara merata beban trafik ke beberapa *server* lain yang terkluster. *Failover* berfungsi untuk meningkatkan ketersediaan yang tinggi (Rosalia et al., 2016).

2.2.3 Load Balancing

Load balancing adalah salah satu teknik atau metode yang digunakan dalam pembagian beban *web server* dalam jaringan. Teknik ini mendistribusikan beban trafik pada dua atau lebih jalur koneksi secara seimbang, agar trafik dapat berjalan optimal. *Load balancing* juga mendistribusikan beban kerja secara merata di dua atau lebih komputer, *link* jaringan, *CPU*, *hard drive*, atau sumber daya lainnya, untuk mendapatkan pemanfaatan *resource* yang optimal (Dewobroto, 2009).

Dengan mempunyai banyak *link* maka optimalisasi utilisasi sumber daya, *throughput*, atau *response time* akan semakin baik karena mempunyai lebih dari satu *link* yang bisa saling mem-*backup* pada saat *network down* dan menjadi cepat pada saat *network* normal jika memerlukan realibilitas tinggi yang memerlukan 100% koneksi *uptime* dan yang menginginkan koneksi *upstream* yang berbeda dan dibuat saling mem-*backup* (Lukitasari & Oklilas, 2013).

Dalam penelitian ini penulis menggunakan algoritme *Round Robin* sebagai algoritme yang akan diimplementasikan pada *load balancing*, karena dengan penggunaan metode tersebut *load balancing* mampu menangani *request* dari *client* untuk membangun session pada *node server* pertama hingga *node server* terakhir. Dan sejalan dengan kebutuhan *shared session* serta konsep dasar *round robin* yang bekerja membagi beban trafik secara bergiliran dan berurutan dari satu *server* ke *server* lain sehingga membentuk putaran sesuai *request client* dari *node server* satu yang telah terbentuk *session* lalu menyebarkan menuju *node server* yang lain. Ini mampu mengurangi kemungkinan resiko *user* kehilangan *session* dengan tanpa membuat *session* baru dan kebutuhan *resource* lebih sedikit sehingga kinerja *server web* lebih seimbang.

2.2.4 Algoritma *Round Robin*

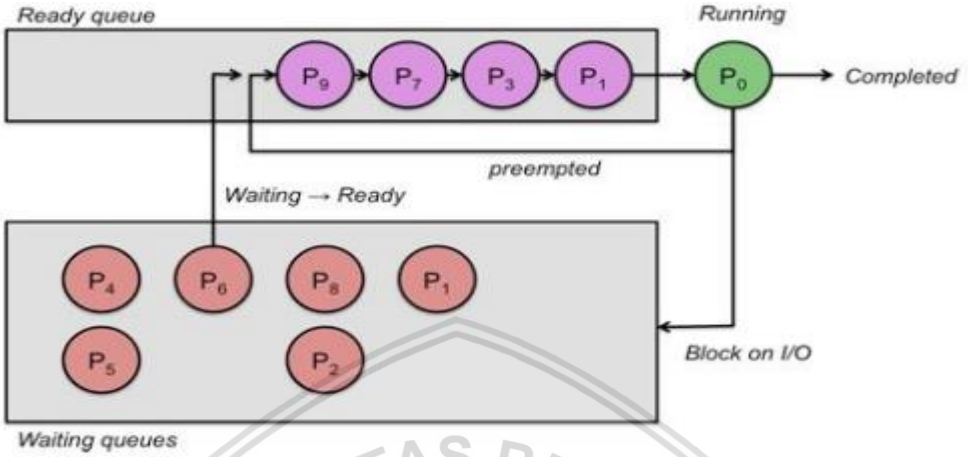
Salah satu algoritma penjadwalan yang akan digunakan dalam penelitian ini adalah *Round Robin*. Setiap proses mendapatkan slot waktu yang disebut dengan *quantum*, yang menginjinkan proses tersebut berjalan untuk selang waktu tertentu. Jika suatu proses tetap butuh berjalan hingga akhir *quantum*, *CPU* akan mendapatkan interupsi untuk menjalankan proses lainnya. Namun, ketika suatu proses telah diblok atau selesai dieksekusi sebelum *quantum* habis, *CPU* melakukan pergantian untuk proses lain. *Scheduler* selalu memeriksa *queue* berisi proses-proses yang siap dieksekusi. Ketika sebuah proses telah menyelesaikan suatu *quantum*, proses tersebut akan ditempatkan pada akhir *queue* (Saputro, 2014). Berikut *pseudo-code* dari algoritma round robin dan proses penjadwalan dapat dilihat pada gambar 2.1.

Algoritma *Round Robin* memiliki *pseudo-code* sebagai berikut :

1. Lakukan pengecekan kosong tidaknya *queue* yang digunakan.
2. Jika *queue* kosong, semua proses ditempatkan ke *queue*.
3. *Process Scheduler* memilih proses pertama.
   a) Atur *timer* sesuai waktu *quantum*
   b) Alokasi sumber daya *CPU*
4. Jika *Burst Time* masih tersisa ketika *quantum* habis :
   a) Proses dihentikan dan ditempatkan pada akhir *queue* (*tail*).
   b) Informasi *Burst Time* disimpan di *PCB (Process Control Block*).
5. Jika *Burst Time* < *Waktu Quantum* :
   a) Proses selesai : alokasi sumber daya dilepaskan dan proses dikembalikan ke *user*.
   b) Interupsi permintaan oleh *I/O* : informasi disimpan di *PCB* dan dihubungkan ke *queue I/O*.

6. Ketika permintaan *I/O* telah terpenuhi :

   a) Proses akan ditempatkan di *tail queue*



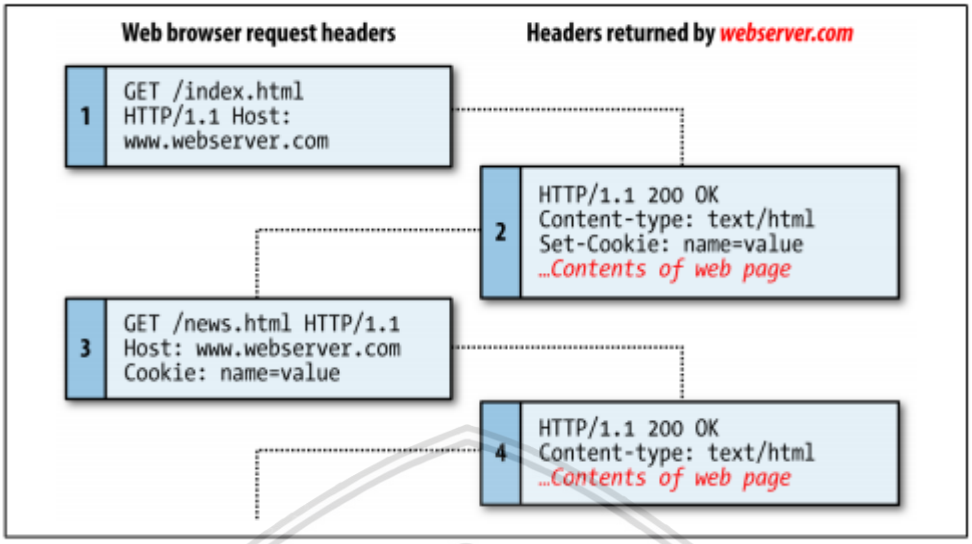
Gambar 2.1 Proses penjadwalan algoritma *Round Robin*

Sumber: (Saputro, 2014)

2.2.5 Cookie

Cookie biasa juga dikenal dengan istilah *HTTP Cookie*, *Web Cookie* maupun juga *browser Cookie*. Apabila dilihat dari letak dan kegunaannya maka *cookie* dapat diartikan sebagai catatan yang digunakan oleh aplikasi *website* untuk mengirimkan informasi berupa status ke *web browser* pengunjung yang digunakan sebagai pengingat *web server* terhadap pengunjungnya (Support Microsoft, 2007). Pada dasarnya prinsip kerja dari komunikasi HTTP adalah sekali *request*, *response* dan selesai. Agar *web server* tetap mengenali kliennya, maka diperlukanlah *cookie*. *Cookie* akan disimpan oleh *browser* dan akan dikirim ulang ketika *browser* melakukan request kembali kepada *web server* (Sasongko, 2014).

Cookie bukanlah bentuk dari perangkat lunak, karena tidak dapat diprogram membawa *virus*, maupun menjalankan file eksekusi (Support Microsoft, 2007). Bahkan *cookie* dapat dicuri untuk mendapatkan akses pada layanan pemilik akun web. Kehadiran *cookie* pada *browser* tidak diketahui oleh penggunanya, karena penyimpanannya tidak ada notifikasi maupun konfirmasi. Dalam pengirimannya dari *web server* ke dalam *browser*, *cookie* berada satu bagian pada *script response* oleh *web server* (Sasongko, 2014). Karena itu, perencanaan penggunaan *cookie* yang cermat sangatlah penting. Maka mengilustrasikan dialog *request* dan *response* yang khas antara *browser web* dan *server* untuk mendapatkan sebuah *cookie*, gambar ilustrasi tersebut dapat dilihat pada gambar 2.2.



Gambar 2.2 Cara kerja *cookie* terhadap *web server* melalui *browser*

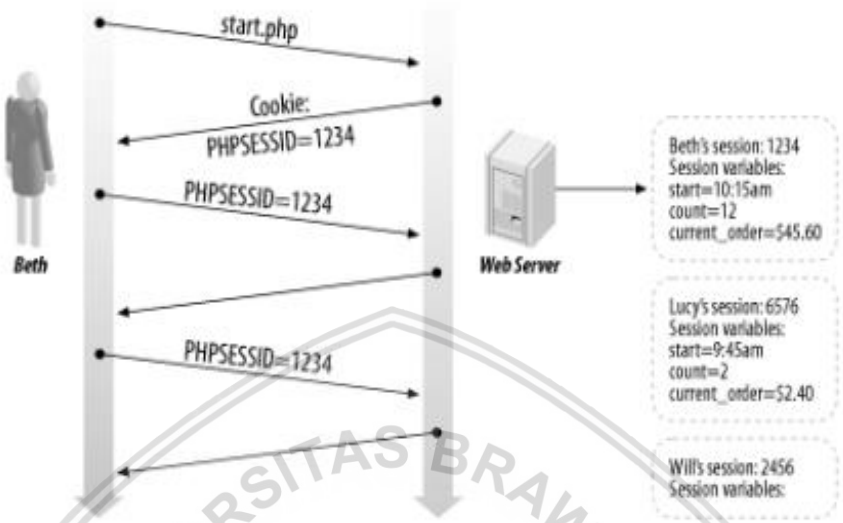
Sumber: (Nixon, 2014)

2.2.6 Session

*Sessio*n merupakan bagian yang cukup penting untuk aplikasi berbasis web. Dengan *session* ini memungkinkan programmer menyimpan informasi user secara semi permanen, artinya selama masa tertentu informasi akan tersimpan. Penyimpanan isi variabel dari *session* yang dibuat disimpan di sisi server, maka tidak bisa diakses secara langsung oleh *client*. Dalam aplikasi berbasis web, umumnya digunakan untuk autentifikasi *login* akun *user*, serta *session* dapat diatur oleh programmer untuk mengatur siapa saja yang berhak mengakses suatu halaman web (Andrian Giurca, 2006).

Session (Sesi) ini memiliki dua komponen, yang terdiri dari variabel *session* dan pengenal *session* (ID). Variabel *session* adalah memuat tentang informasi dasar yang terkait dengan interaksi pengguna dengan aplikasi. Misalnya, variabel *session* tersebut menyimpan dalam data aktifitas pengguna yakni berisi 5 item, siapa nama pengguna, waktu pengguna aktif *login*, *item* apa yang dilihat oleh pengguna dan pengguna *login* ke dalam aplikasi. Variabel *session* disimpan di *server web* atau *server* basis data, dan disimpan menggunakan *session* ID. Lalu saat *session* dimulai, *user* yang aktif pada *browser* akan diberikan ID *session*. ID *session* ini kemudian disertakan dengan permintaan berikutnya ke *server*. Fungsi berguna agar saat browser membuat permintaan ke sebuah *server* menggunakan ID *session* untuk menemukan variabel *session* yang sesuai, dan variabel dibaca atau ditulis sesuai kebutuhan. Dalam prakteknya, variabel *session* biasanya disimpan di sisi *server web* dalam *file* bawaannya berupa *PHP file* (Andrian Giurca, 2006). Berikut cara kerja dari sebuah

session yang umumnya digunakan pada sebuah *website* yang dapat dilihat pada gambar 2.3.



Gambar 2.3 Cara kerja *session* melalui *web server*

Sumber: (Giurca, 2006)

Ada tiga karakteristik dalam mengelola *session* pada sebuah *web*:

1. Informasi atau keadaan harus disimpan. Informasi yang harus dipertahankan di beberapa permintaan *HTTP* disimpan dalam variabel sesi.
2. Setiap permintaan *HTTP* harus membawa identitas yang memungkinkan *server* untuk memproses permintaan dengan variabel sesi yang benar.
3. Sesi harus memiliki batas waktu. Kalau tidak, jika pengguna meninggalkan situs *web* maka akan memungkinkan *server* tidak dapat mengetahui kapan sesi harus berakhir.

2.2.7 Shared Session

Shared session ini digunakan untuk memindahkan atau melemparkan *session* pada *server* pertama sebelumnya ke *server* yang lain, agar saat dipindahkan tidak kehilangan *session* sehingga membuat *user* harus membuat *session* yang baru (Dwi Prastyo, 2017).

Mekanisme dalam *shared session* ini dapat mengadaptasikan dua metode alternatif penyimpanan yakni menggunakan *Memory cache* atau *Database MySQL*. Dalam penelitian ini menggunakan *Database MySQL* serta *PHP* untuk merancang *shared session*. Sebelum masuk dalam tahap *shared session,* data *session* yang dibuat perlu disimpan terlebih dahulu ke dalam *server database* dengan cara mebuat kode

program melalui fungsi ***CREATE*** pada tabel *database pesbuk* seperti berikut dapat dilihat pada tabel 2.2.

Tabel 2.2 *Syntax* membuat tabel "*session*" pada *Database MySQL*

```
1  CREATE TABLE 'ci_sessions' (
2      'id' varchar(40) NOT NULL,
3      'ip_address' varchar(45) NOT NULL,
4      'timestamp' int(10) UNSIGNED NOT NULL DEFAULT '0',
5      'data' blob NOT NULL
6  ) ENGINE=InnnoDB DEFAULT CHARSET=latin1;
```

Kemudian dalam *PHP* melalui *library* bawaan CodeIgniter menyediakan fungsi yang memungkinkan untuk mewakilkan mekanisme *session* bawaan dengan menentukan nama-nama fungsi tersendiri dalam menangani setiap tugas-tugas yang berbeda. Fungsi ini disebut *session_set_save_handler*(), dan tiap fungsi berjumlah enam argumen. Dari setiap fungsi-fungsi ini memiliki tanggung jawab tugas seperti, berikut:

1) *Opening the session data store*
2) *Closing the session data store*
3) *Reading session data*
4) *Writing session data*
5) *Destroying all session data*
6) *Cleaning out old session data*

Dapat diasumsikan dari keenam argumen yang dibutuhkan dengan kode program seperti berikut dapat dilihat pada tabel 2.3.

Tabel 2.3 *Source code* fungsi *handler session* pada *library CodeIgniter*

```
1  defined('BASEPATH') OR exit('No direct script access allowed');
2  interface SessionHandlerInterface {
3         public function open($save_path, $name);
4         public function close();
5         public function read($session_id);
6         public function write($session_id, $session_data);
7         public function destroy($session_id);
8         public function gc($maxlifetime);
9  }
```

Komponen utama yang harus dipanggil yakni *session_set_save_handler*() sebelum memanggil *session_start*(), tetapi tetap menentukan fungsi itu sendiri. Fungsi-fungsi tersebut telah dikonfigurasi secara bawaan oleh *PHP* di dalam *library CodeIgniter* saat melakukan perancangan sebuah *website* sebagai perantara pengujian tersebut, semua fungsi data yang berkaitan mengenai *session handler* akan dipanggil.

2.2.8 PHP

PHP yaitu singkatan dari "PHP: Hypertext Prepocessor", PHP adalah bahasa scripting yang menyatu dengan *HTML* dan dijalankan pada *server side*. Artinya semua sintaks yang kita berikan akan sepenuhnya dijalankan pada *server* sedangkan yang dikirimkan ke browser hanya hasilnya saja (Widigdo, 2012). Pembuatan *web* ini kombinasi antara *PHP* sendiri sebagai bahasa pemrograman dan *HTML* sebagai pembangun halaman *web* (Sunarfrihantono, 2002).

File PHP dapat berisi *teks*, *tag HTML*, dan *Script*. *File PHP* tersebut dikembalikan ke *browser* dalam bentuk *plain HTML*. *File PHP* ini dapat berekstensi *.php, .php3* atau *.phtml*. Kelebihan *PHP* sebagai bahasa *script* yakni dapat dijalankan pada berbagai platform sistem operasi seperti *Windows*, *Linux*, dll. Serta kompatibel terhadap hampir semua *server* yang digunakan saat ini dan bebas diunduh dari situs resmi *PHP* (Erawan, 2014).

2.2.9 Database MySQL

SQL (Structured Query Language) adalah bahasa standart yang digunakan untuk mengakses *server database*. Semenjak tahun 70-an bahasa ini telah dikembangkan oleh IBM, yang kemudian diikuti dengan adanya *Oracle, Informix* dan *Sybase*. Dengan menggunakan *SQL*, proses akses database menjadi lebih user-friendly dibandingkan dengan misalnya *dBase* ataupun *Clipper* yang masih menggunakan perintah-perintah pemrograman murni.

MySQL adalah sebuah *database management system* (DBMS) popular yang memiliki fungsi sebagai relational *database management system* (RDBMS). Selain itu *MySQL software* merupakan suatu aplikasi yang sifatnya open source serta server basis data *MySQL* memiliki kinerja sangat cepat, *reliable*, dan mudah untuk digunakan serta bekerja dengan arsitektur *client server* atau *embedded systems* (MySQL, 2012). Dikarenakan faktor *open source* dan popular tersebut maka cocok untuk mendemontrasikan proses replikasi basis data.

2.3 Komponen Pendukung

2.3.1 Mesin Virtual

Sebuah mesin virtual atau mesin maya (VM) adalah implementasi perangkat lunak dari sebuah mesin (misalnya komputer) yang mengeksekusi program-program seperti mesin fisik. Mesin virtual dibedakan menjadi dua kategori utama, didasarkan pada penggunaan dan tingkat korespondensi untuk setiap mesin nyata. Sebuah mesin virtual sistem menyediakan lengkap platform sistem yang mendukung pelaksanaan lengkap sistem operasi (OS). Sebaliknya, mesin virtual proses didesain untuk menjalankan satu program, yang berarti bahwa ia mendukung satu proses Karakteristik penting dari sebuah mesin virtual adalah bahwa perangkat lunak yang berjalan di dalam terbatas pada sumber daya dan abstraksi yang disediakan oleh

mesin virtual tidak dapat keluar dari dunia virtual (Ardhitya, 2007). Secara umum terdapat dua jenis *virtual machine*, yaitu:

a) *Virtual Machine* Aplikasi, adalah jenis *virtual machine* yang dapat menjalankan aplikasi di atas sistem operasi. Biasanya sering disebut sebagai *middleware* karena bekerja diantara sistem operasi dan aplikasi komputer. Contoh dari *virtual machine* ini adalah *Java Virtual Machine* dan *Common Language Runtime*.

b) *Virtual Machine* Sistem Operasi, adalah jenis *virtual machine* yang dapat menciptakan lingkungan sistem komputer atau sering disebut sebagai komputer virtual agar dapat menjalankan sistem operasi yang lain. *Virtual machine* ini sering disebut sebagai emulator, karena mengemulasi sistem operasi menjadi sebuah mesin virtual. Contoh dari mesin virtual ini adalah *VMWare Workstation* dan *Microsoft Virtual PC*.

2.3.2 Oracle VM VirtualBox

Oracle VM VirtualBox adalah perangkat lunak virtualisasi, yang dapat digunakan untuk mengeksekusi sistem operasi "tambahan" di dalam sistem operasi "utama". Sebagai contoh, jika seseorang mempunyai sistem operasi *Microsoft Windows* yang terpasang di komputernya, maka seseorang tersebut dapat pula menjalankan sistem operasi lain yang diinginkan di dalam sistem operasi *Microsoft Windows*. Fungsi ini sangat penting jika seseorang ingin melakukan ujicoba dan simulasi instalasi suatu sistem tanpa harus kehilangan sistem yang ada (Safitri, 2012).

2.3.3 NGINX

NGINX yaitu *server HTTP* dan *Proxy* dengan kode sumber terbuka serta bisa berfungsi sebagai *proxy IMAP/POP3*. Kode sumber *nginx* ini ditulis oleh seorang warga negara Rusia yang bernama Igor Sysoev pada tahun 2002 dan dirilis ke publik pada tahun 2004. *Nginx* terkenal karena stabil, memiliki tingkat performansi tinggi dan minim mengonsumsi sumber daya (NGIX Software.INC, 2014). Selain itu juga *NGINX* dapat digunakan sebagai *load balancing* karena memiliki fitur *proxy pass* yang berfungsi untuk memindahkan jalur ke node yang ditunjuk.

2.3.4 Apache

Apache merupakan *web server* yang paling banyak dipergunakan di *internet*. Program ini pertama kali didesain untuk sistem operasi *UNIX*. Namun demikian, pada beberapa versi berikutnya *Apache* mengeluarkan program versi selanjutnya yang dapat dijalankan di *Windows NT*. *Apache* memiliki program pendukung yang cukup banyak. Hal ini memberikan layanan yang cukup lengkap bagi penggunanya.

Apache yakni sebuah nama *web server* yang bertanggung jawab pada *request-response HTTP* dan *logging* informasi secara detail. Selain itu, *Apache* juga diartikan

sebagai suatu *web server* yang kompak, modular, mengikuti standar protokol *HTTP*, dan tentu saja sangat digemari (Silitonga, Suswaini, & Kurniawan, n.d.).

2.3.5 CodeIgniter

CodeIgniter adalah *framework* aplikasi web yang open source untuk bahasa pemrograman PHP. *CodeIgniter* memiliki banyak fitur yang membuatnya berbeda dengan *framework* lainnya. Tidak seperti beberapa *framework* PHP lainnya, dokumentasi untuk *framework* ini sangat lengkap, yang mencakup seluruh aspek dalam *framework*. CodeIgniter juga mampu berjalan pada lingkungan *shared hosting* karena memiliki ukuran yang sangat kecil, namun memiliki kinerja yang sangat luar biasa. (Griffiths, 2010)

Dari sisi pemrograman, CodeIgniter kompatibel dengan PHP4 dan PHP5, sehingga akan berjalan dengan baik pada *web host* yang banyak dipakai pada saat ini. *CodeIgniter* menggunakan pola desain *Model View Controller* (MVC), yang merupakan cara untuk mengatur aplikasi web ke dalam tiga bagian yang berbeda, yaitu *Model* yang memiliki peran sebagai lapisan abstraksi *database*, *Views* yang memiliki peran sebagai file-file template tampilan depan, dan *Controller* yang memiliki peran sebagai logika bisnis dari aplikasi. Pada intinya, *CodeIgniter* juga membuat penggunakan ekstensif dari pola desain Singleton. Maksudnya adalah cara untuk me-*load class* sehingga jika *class* itu dipanggil dalam beberapa kali, kejadian yang sama pada *class* tersebut akan digunakan kembali. Hal ini sangat berguna dalam koneksi *database*, karena kita hanya ingin menggunakan satu koneksi setiap kali *class* itu digunakan. (Griffiths, 2010)

CodeIgniter dikembangkan oleh Rick Ellis, dengan versi awal yang dirilis pada tanggal 28 Februari 2006. Dari tahun itulah hingga sekarang, telah muncul banyak versi *CodeIgniter* yang terus berkembang dengan penambahan fitur baru dari versi sebelumnya. Untuk versi terbaru dari *codeigniter* adalah versi 2.0.3. (Basuki, 2010)

2.3.6 Black Box testing

Black Box Testing adalah pengujian yang dilakukan hanya mengamati hasil eksekusi melalui data uji dan memeriksa fungsional dari perangkat lunak. Secara harfiah *black box testing* berarti mengevaluasi sistem hanya dari tampilan luar atau *interface*nya dan fungsionalnya tanpa mengetahui apa sesungguhnya yang terjadi dalam proses detailnya. (Reza Palevi, 2013)

2.3.7 Response time

Response time (waktu tanggap) pada sistem interaktif memiliki definisi sebagai waktu yang dihabiskan dari saat karakter terakhir dari perintah dimasukkan atau transaksi sampai hasil pertama muncul di layar. Waktu tanggap ini disebut terminal response time. (Kurniawan, 2012)

Response time server web adalah waktu rata-rata yang dihasilkan pada setiap *server web,* dalam menyelesaikan *request* dari perintah yang dimasukkan setiap detiknya pada saat sistem dijalankan. Secara matematis, kalkuasi perhitungan rata-rata *response time server web* dapat dirumuskan sebagai berikut:

$$\bar{X} = \frac{\Sigma \text{ rt}}{\Sigma \text{ sw}} \quad \textbf{(2.1)}$$

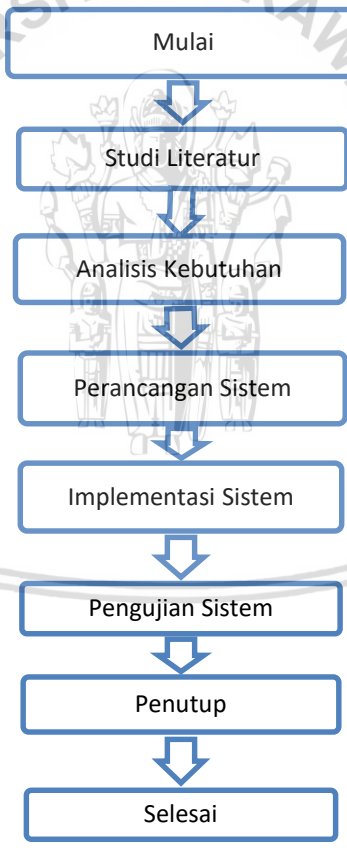
$\bar{X}$ = Total rata-rata hasil *response time* dari *server web*

Σ rt = Jumlah keseluruhan dari *response time* dalam satu klaster *server*

Σ sw = Jumlah dari *server web* yang aktif dalam satu klaster *server*

BAB 3 METODOLOGI

Pada bab ini akan memaparkan sistematika dalam penulisan skripsi, dimulai dari tahap pertama hingga keseluruhan penelitian. Dalam penelitian ini termasuk dalam penelitian implementatif, yang mana bisa diartikan bahwa penelitian ini akan menerapkan sebuah sistem yang tersusun secara matang dan terperinci bedasarkan dasar teori yang dikumpulkan sebelumnya dan memanfaatkan fungsi konseptual yang terencana sekaligus dapat diuji seberapa jauh sistem ini mampu untuk dikembangkan lebih lanjut. Kemudian akan menjelaskan bagaimana rangkaian penyusunan sistem menggunakan literatur yang sudah direncanakan. Setelah sistem yang diimplementasikan selesai, lalu masuk dalam proses pengujian sistem dengan melakukan verifikasi dan validasi data berdasarkan sistem yang sudah dibangun dan dirancangkan. Diagram alir metodologi penilitian yang dilakukan dapat dilihat pada gambar 3.1.



Gambar 3.1 Diagram Alir Metodologi Penilitian

3.1 Studi Literatur

Studi literatur mempelajari mengenai penjelasan dasar teori yang digunakan untuk menunjang penulisan skripsi. Teori-teori pendukung tersebut diperoleh dari buku, artikel, jurnal, *E-book*, dan dokumentasi *project*. Teori-teori pendukung tersebut meliputi :

1. Penelitian-penelitian terkait;
2. *Web Server*;
3. *Cookies* dan *Sessions*;
4. *Shared Session*;
5. *Load Balancer Round Robin*;
6. *PHP* dan *MySQL*;

3.2 Analisis Kebutuhan

Analisa kebutuhan digunakan agar mengetahui apa saja yang kebutuhan-kebutuhan yang diperlukan dalam penelitian sebelum diimplementasikan, meliputi:

3.2.1 Kebutuhan Sistem *Share Session*

Pada bagian ini menjelaskan kebutuhan *sistem share session* yang ditampilkan melalui *website*. Karena tujuan dari sistem *shared session* ini dibuat yakni dapat diimplementasikan pada *server web* kemudian dapat ditampilkan pada halaman *website*. Berikut beberapa daftar kebutuhan sistem yang harus dipenuhi agar sesuai dengan tujuan sistem ini dibuat. Kebutuhan sistem *share session* pada *website* berikut akan dijabarkan secara lebih lanjut pada kebutuhan fungsional sistem dan kebutuhan pendukung sistem.

3.2.2 Kebutuhan Fungsional Sistem

Kebutuhan pada sistem yang digunakan untuk penelitian ini yaitu:

1. *Virtualbox* dapat memberikan akses serta layanan untuk *server load balancing* dan melakukan mekanisme algoritme *Round Robin* untuk mendistribusikan trafik ke seluruh *node server* yang aktif.
2. Aplikasi *web server* dapat di replika pada 4 *node server*.
3. Keempat *web server* dapat menyediakan layanan berbasis data yang ditampilkan pada halaman *website*, meliputi:
   a) *Website* dapat menampilkan fitur halaman beranda.
   b) *Website* dapat menampilkan fitur halaman *login* dan *logout.*
   c) *Website* dapat menampilkan fitur halaman *register.*

d) *Website* dapat menampilkan fitur halaman *timeline.*

4. *Web server* dapat mengambil dan menyimpan data di *database.*
5. *Web server* dapat membuat data *session* dan disimpan pada d*atabase* .
6. *Web server* dapat mengirimkan data *session* untuk disimpan ke *database*
7. *Web server* dapat menerima data *session* yang telah tersimpan *database*

3.2.3 Kebutuhan Pendukung Sistem

Analisis kebutuhan pada sub bab ini menjelaskan kebutuhan yang akan digunakan mendukung tahap implementasi yang akan dilakukan. Kebutuhan pendukung sistem dalam menunjang mekanisme sistem *shared session* pada klaster *server web.* Analisis kebutuhan mencakup kebutuhan perangkat keras dan kebutuhan perangkat lunak.

3.2.3.1 Load Balancing

Aplikasi *load balancing* dibutuhkan untuk mendistribusikan jalur trafik secara seimbang agar trafik yang diminta berjalan secara optimal dan efisien. Karena tidak mengimplementasikan perangkat secara *hardware*, *node server load balancing* dalam penelitian ini dibuat dengan menginstalkan aplikasi *load balancing NGINX*. Berikut ini adalah kebutuhan fungsional aplikasi *NGINX* yang diperlukan pada tabel 3.1.

Tabel 3.1 Kebutuhan Pendukung *Load balancing*

No.	Kebutuhan Pendukung
1	Aplikasi Nginx dapat terkoneksi dan mendistribusikan trafik ke dalam 4 server web
2	Aplikasi Nginx dapat membagi jalur trafik dengan melakukan mekanisme algoritma Round-Robin
3	Aplikasi Nginx dapat terkoneksi ke database.

3.2.3.2 Web Server

Aplikasi *web server* dibutuhkan untuk bertanggung jawab dalam menangani *request* dan *response* HTTP sekaligus *log* informasi data secara mendetail. Aplikasi yang digunakan pada *web server* dalam penelitian ini ialah *Apache*. Berikut ini adalah kebutuhan pendukung aplikasi *Apache* yang diperlukan pada tabel 3.2.

Tabel 3.2 Kebutuhan Pendukung *Web Server Apache*

No.	Kebutuhan Pendukung
1	Aplikasi web server apache dapat terkoneksi ke load balancing
2	Aplikasi web server apache dapat menerima request dan respon yang dikirim oleh load balancer
3	Aplikasi web server apache dapat terkoneksi ke database.
4	Aplikasi web server apache dapat meminta dan menampilkan data dari database.

3.2.3.3 Database

Aplikasi *database* dibutuhkan untuk bertanggung jawab dalam menangani sistem manajemen data yang mampu mengolah, menampung dan maniplulasi data. Aplikasi yang digunakan pada *database* dalam penelitian ini ialah *MySQL*. Berikut ini adalah kebutuhan fungsional aplikasi *MySQL* yang diperlukan pada tabel 3.3.

Tabel 3.3 Kebutuhan Pendukung *Database MySQL*

No.	Kebutuhan Pendukung
1	Aplikasi database mysql dapat terkoneksi ke 4 node *server web* yang aktif
2	Aplikasi database mysql dapat menngirimkan data *session* ke 4 node *server web* yang aktif
3	Aplikasi database mysql dapat menyimpan data *session* dari 4 node *server web* yang aktif
4	Aplikasi database mysql dapat menerima data *session* dari 4 node *server web* yang aktif

3.2.4 Kebutuhan Perangkat Keras

Menjelaskan spesifikasi dari perangkat keras yang digunakan dalam pengujian penelitian skripsi ini. Berikut kebutuhan perangkat keras laptop yang dibutuhakan pada tabel 3.1.

Tabel 3.4 Kebutuhan Perangkat Keras

Nama Perangkat	Spesifikasi Perangkat
Laptop	ASUS A450C
	RAM 8 GB
	Procesor Intel Core i3
	Hardisk 500 GB

3.2.5 Kebutuhan Perangkat Lunak

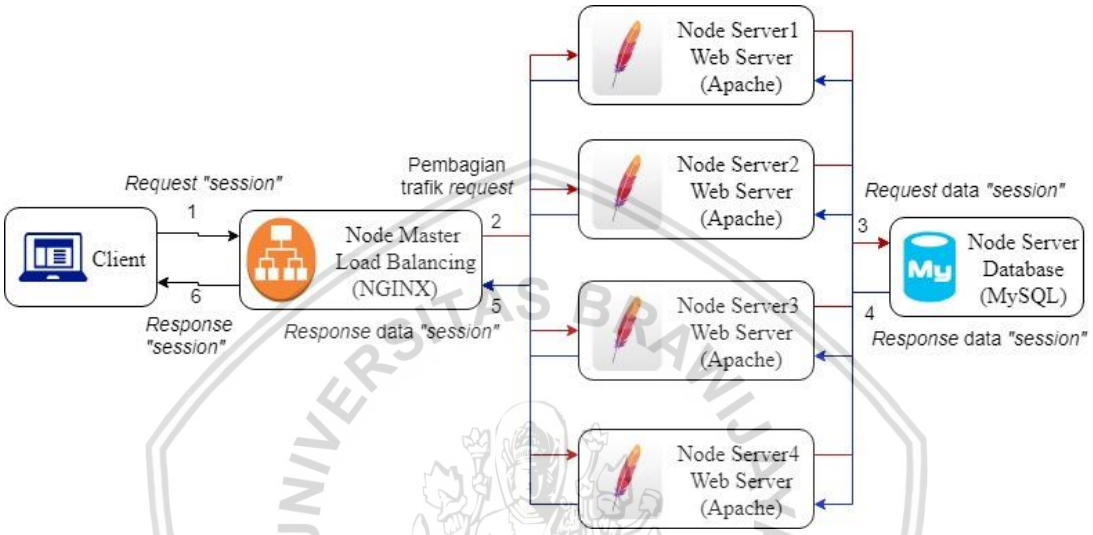
Menjelaskan spesifikasi dari perangkat lunak yang digunakan dalam pengujian penelitian skripsi ini. Berikut kebutuhan perangkat lunak yang digunakan:

Tabel 3.5 Kebutuhan Perangkat Lunak

Nama Perangkat	Spesifikasi Perangkat
Virtual Machine	Oracle VM Virtual Box
	Sistem Operasi: Linux Ubuntu Server 18.04 64-bit
	Kernel: Ubuntu 18.04 Bionic Beaver 4.15.0-10-generic
	Jumlah Seluruh Node: 5 Node Server
	Memori: 600 Mb/Node
Sistem Operasi Host	Linux Ubuntu 18.04 64-bit
Bahasa Pemograman	PHP
Framework	CodeIgniter
Database	MySQL
Aplikasi Pengujian	JMeter

3.3 Perancangan Sistem

Perancangan sistem dilakukan setelah studi literatur dan semua kebutuhan sistem yang akan digunakan telah diperoleh melalui tahapan analisis kebutuhan. Untuk penjelasan rancangan arsitektur sistem secara lebih detail dapat dilihat pada bab selanjutnya dalam sub bab alur komunikasi sistem. Rancangan arsitektur sistem dapat dilihat pada gambar 3.2



Gambar 3.2 Rancangan Arsitektur Sistem

Berikut ini adalah keterangan rancangan sistem berdasarkan perangkat yang digunakan dalam membangun sistem:

1. *Load Balancing NGINX*

   Load balancing yang akan membagi trafik ke *web server* yang mana untuk membentuk sebuah *session* baru sebagai tanda pengenal untuk *user* yang telah melakukan *login*. Keempat *web server* yang ditangani oleh *load balancing* memiliki fungsi dan peran yang sama ataupun bisa sebaliknya karena *load balancing* menggunakan mekanisme *Round-Robin*.

2. Klaster *Web Server*

   Pada klaster *web server* ini terdapat empat *web server* aktif yang masing-masing memiliki peran dapat menerima data *session* dan mengirim *session* kemudian diarahkan oleh *load balancing* untuk menyimpan *session* yang telah dibuat tersebut kedalam database MySQL. Selanjutnya bagian *web server* 2, *web server* 3 ataupun *web server* 4 memiliki fungsi yang tidak jauh berbeda dengan *web server* 1 sekaligus juga bisa bertindak sebagai *web server* yang mampu meng*handle* jika *web server* 1 tidak dapat berjalan atau terjadi kegagalan *service* dalam menampilkan *website* yang akan dirancang. Jadi *web*

server lainnya dapat meminta data *session* yang sebelumnya sudah dibuat dan disimpan pada *database*.

3. *Database MySQL*

   Pada bagian *database MySQL* berfungsi sebagai pusat penyimpanan data utama dan termasuk utamanya mampu menyimpan data *session* yang dibuat oleh keempat *web server*.

3.4 Implementasi Sistem

Implementasi sistem disusun berdasarkan perancangan kebutuhan dan perancangan sistem yang telah dibuat sebelumnya. Adapun implementasi sistem sebagai berikut.

1. Melakukan konfigurasi *Virtualbox* dan *Linux Ubuntu Server*
2. Memasang *Load Balancing*
3. Melakukan konfigurasi aplikasi pada klaster *Web Server*
4. Melakukan konfigurasi sistem pada *database MySQL*
5. Melakukan penerapan *Shared Session* melalui *PHP CodeIgniter*
6. Mengakses *Web Server* melalui aplikasi *website* oleh *client*
7. Menganalisa hasil *output*

3.5 Pengujian Sistem

Dalam tahapan pengujian, peneliti melakukan pengujian kinerja *session* terhadap jumlah *request* yang dikirim oleh *client*. Dengan menggunakan alat uji berupa *JMeter*, alat uji tersebut digunakan untuk bertujuan mengirim permintaan *session* yang telah tersimpan kepada *database* untuk kebutuhan autentikasi *login*. *Load balancer* sendiri akan *forwarding* permintaan tersebut ke *server web* 1 maupun *server web* 2 secara berurutan dan sebaliknya. Namun lebih detail dalam penelitian ini terdapat 2 skema pengujian untuk menguji metode *shared session* secara lebih terperinci akan dijabarkan pada bab perancangan pengujian. Masing-masing skenario akan dijalankan pada sistem yang sudah dirancang, dengan tujuan untuk mengetahui apakah sistem bisa dijalankan pada skenario-skenario yang telah dibuat. Dan pada pengujian kinerja tersebut dilakukan untuk mengetahui hasil kinerja dari aplikasi yang telah dirancang. Parameter keberhasilan sistem yang digunakan yaitu *latency, response time, connection error* dan *connection success*.

3.6 Penutup

Pada bab penutup akan berisi kesimpulan dan saran dari penulis. Pengambilan kesimpulan dapat dibuat setelah perancangan, implementasi, pengujian selesai

dilakukan. Kesimpulan disusun bedasarkan hasil dari pengujian tehadap penelitian yang dilakukan. Isi dari kesimpulan tersebut diharapkan mampu memberikan hasil terbaik dari rumusan masalah yang diajukan diantaranya yakni merancang dan mengimplementasikan arsitektur *shared session* pada *web server*.

Pada akhir penulisan ini adalah saran yang memiliki tujuan untuk memperbaiki dan menyempurnakan terhadap skripsi yang telah dibuat serta acuan untuk pengembangan lebih lanjut.

BAB 4 PERANCANGAN

Pada bab ini bertujuan untuk menjabarkan perancangan dari sistem *shared session* yang akan dibangun pada klaster *server web*. Adapun kebutuhan yang dijabarkan adalah kebutuhan yang diperlukan untuk membangun *sistem load balancing* pada *Virtual Machine* dengan berpedoman pada metodologi penelitian yang telah dibahas.

4.1 Ruang Lingkup

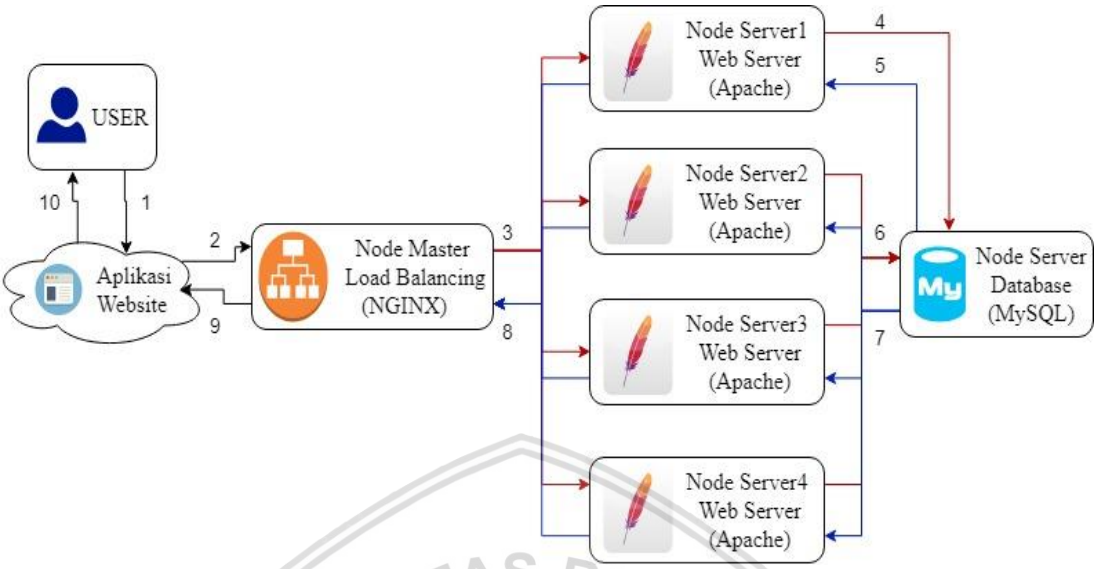
Sistem yang dibangun merupakan sebuah sistem yang menggabungkan beberapa *node server* melalui *Virtual Machine*. Terdapat lima *node server* yang dikelola oleh *VirtualBox*. Pada empat node web server dilakukan *deploying service* dengan menggunakan aplikasi *Apache*. Selain itu, *load balancing* yang akan digunakan juga memanfaatkan *image Nginx*. Salah satu *node server* yang dibuat *node* yang sama sebagai *load balancer* dan sekaligus *database server* karena keterbatasan ukuran ruang memori dari sistem operasi *Linux*.

4.2 Perancangan Sistem

Perancangan dilakukan untuk melakukan perencanaan terhadap sistem yang dibangun berdasarkan kebutuhan yang telah dispesifikasikan.

4.2.1 Alur Komunikasi Sistem *Shared Session*

Pada sistem ini ada tiga bagian utama yang menjalankan tugasnya masing-masing, yaitu *Load balancing* menjadi pendistribusian beban trafik, *web server* 1, 2, 3 dan 4 yang menjadi aplikasi yang bertanggung jawab menyediakan layanan berbasis data berupa *Apache* dan pusat sekaligus penyimpanan data berbasis *database MySQL* sebagai tujuan untuk disimpan dan dibagikan ke seluruh *web server*. Untuk alur pada rancangan ini merupakan gambaran secara umum yang ditampilkan melalui 5 *node server*. Perancangan alur komunikasi sistem dapat dilihat pada gambar 4.1.



Gambar 4.1 Alur Komunikasi Sistem *Shared Session*

Berikut ini adalah penjelasan terperinci bedasarkan rancangan dari alur komunikasi sistem:

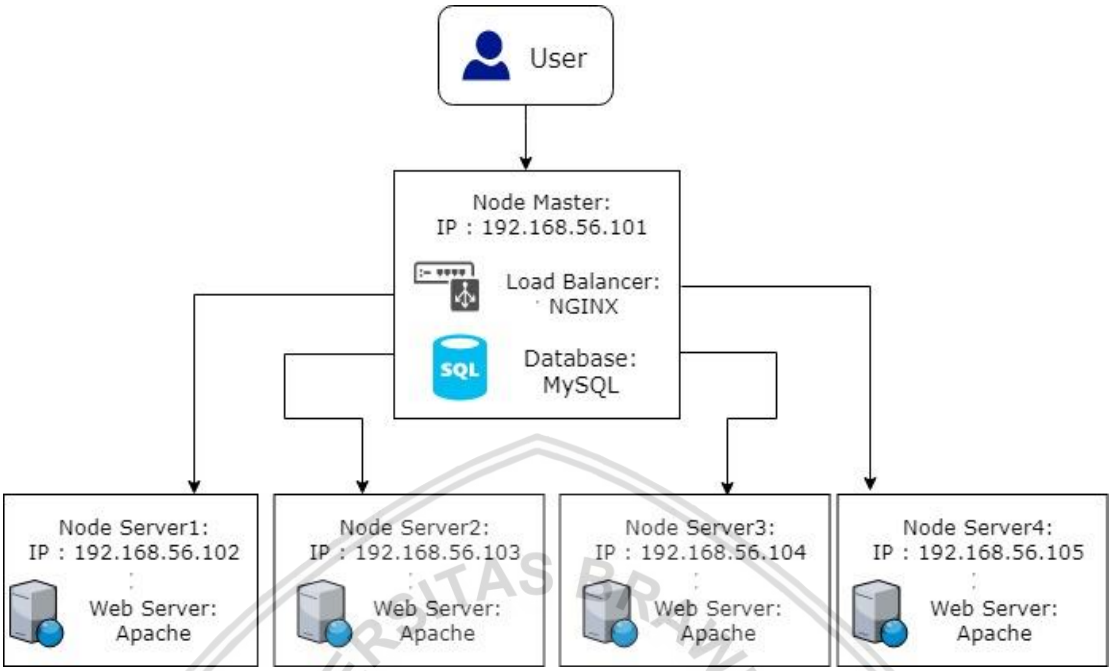
1. *Login,* suatu proses yang dilakukan oleh *user* untuk memasukkan identitas akun yang sudah terdaftar melalui *form* halaman *login* pada aplikasi *website.* Proses *request* tersebut diarahkan ke salah satu *web server* melalui aplikasi *load balancing*.
2. Selanjutnya pada saat melakukan proses *login,* sistem akan melakukan validasi data dari *username* dan *pasword* yang sebelumnya telah didaftarkan oleh *user* pada aplikasi *website*.
3. Pada salah satu *web server* yang menerima *request* dari *user* setelah diarahkan oleh *load balancing* menggunakan algoritme *Round Robin*, kemudian akan membentuk sebuah *session* baru bedasarkan *session ID* yang didapatkan dari data *cookies* di browser sebagai tanda pengenal dalam bentuk data bahwa *user* yang sudah berhasil melakukan proses *login*.
4. Kemudian *session* baru tersebut akan ditandai untuk disimpan di *database MySQL* dalam bentuk data *string* pada tabel "*ci_sessions*". Lalu dalam bentuk permintaan akan dikirim ke *database* bahwa *session* saat itu telah *login* dan sudah tersimpan pada *database*.
5. Mengembalikan data yang sebelumnya telah di *request* yakni *session*, berupa *response* data untuk *user* bahwa *session* yang dibentuk sudah disimpan.
6. Saat setelah *session* dapat tersimpan di *database* dalam beberapa waktu tertentu ada kondisi *shared session* ini akan aktif, yakni jika salah satu *web server* meminta akses *session* yang sama, misalnya diasumsikan salah satu *server web* mati dari keempat *server web*. Akhirnya *server web* yang aktif

tersebut memerlukan data *session* yang sudah tersimpan untuk dibagikan ke *web server* lainnya yang aktif. Dengan diatasi oleh *server web* yang aktif, dengan cara *server web* tersebut memeriksa data *user* dan menyesuaikan dengan *session ID* yang aktif pada *database*. Secara otomatis kemudian user tidak perlu melakukan autenfitikasi *login* yang sama meskipun *user* tersebut menjalankan proses *login* pada *web server* yang mati atau mengalami kegagalan menyediakan *service*.

7. Setelah selesai mengambil data *user* dan mencocokan *session ID* yang sama kemudian sistem secara otomatis memproses untuk melakukan *login* otomatis agar data *user* bisa diakses pada *web server* aktif yang lainnya. Maka *user* dapat melajutkan aktifitas pada halaman *website* tersebut tanpa terhambat kendala *login* kembali.
8. *Web server* yang aktif tersebut, yang ditampilkan melalui aplikasi *website* mengirimkan respon ke *user* melalui perantara *load balancing* bahwa *session* saat itu telah aktif dan berhasil di akses.
9. *Login* berhasil secara otomatis dilakukan sistem dimana merupakan tanda proses terakhir bahwa seluruh *session* dapat dibagikan ke seluruh *web server* yang membutuhkan akses *session* tersebut.
10. Menampilkan halaman *timeline* bahwa *user* dapat masuk kedalam aplikasi *website* walaupun mengakses *session* dari *server web* yang berbeda dalam satu klaster.

4.2.2 Topologi klaster *server web*

Peneliti menggunakan lima *node server* yaitu satu *node* sebagai *master* dan empat *node* sebagai *server web* yang aktif dalam satu klaster *server web*. *Node master* adalah tempat perintah *load balancer* dan *database MySQL* dijalankan. Load balancer yang bertugas menangani penjadwalan serta pencarian layanan *DNS* dan *database* sebagai pusat menyimpan seluruh data (diwakilkan dalam gambar oleh kotak kecil) pada semua *node server*. Tampilan topologi klaster *server web* dapat dilihat pada gambar 4.2.



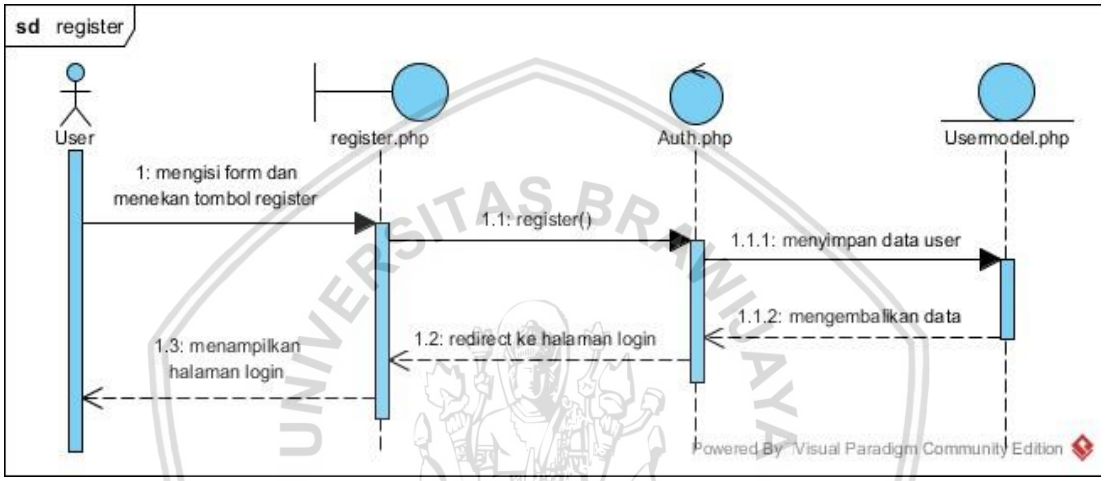
Gambar 4.2 Topologi klaster *server web*

Load balancing dan *Database* pada *Virtualbox* ini berjalan di *node master* dengan *IP server* 192.168.56.101, dimana *Load balancing* mempunyai peran yang dapat memproses *request user* yang masuk dan membagi trafik *request* yang telah dibuat oleh *server web* yang aktif, lalu meneruskan *request* tersebut masuk ke seluruh *node server* yang telah ditentukan. Perancangan sistem ini membutuhkan koneksi jaringan IP lokal agar masing-masing komponen atau *node* saling terhubung. Proses diawali dengan membuat suatu konten atau layanan pada *server master,* lalu secara otomatis konten atau layanan yang telah dibuat akan tersebar ke semua *node server* yang tergabung kedalam satu klaster *web server*. Kemudian sistem mengaktifkan konfigurasi *load balancing* menggunakan algoritme *Round-Robin* pada *server master* untuk membagi beban *request.* Kemudian *client* dapat mengakses layanan atau konten yang sudah dibuat pada sistem tersebut. *Database MySQL* ditempatkan pada satu *node server* yang sama dengan *load balancer*, *database* berfungsi menyimpan data *session* yang dibangun oleh salah satu *server web*. Jika sewaktu-waktu *server web* lainnya yang membutuhkan session namun belum mendapatkan akses *session* dari *user* yang sama maka dapat meminta langsung ke *database* yang telah menyimpan cadangan data *session*.

4.2.3 Perancangan Diagram *Sequence*

4.2.3.1 Diagram *Sequence Register*

Alur diagram *sequence* ini menjelaskan alur proses *register* dari website aplikasi yang bertujuan agar *user* dapat mendaftarkan identitas dirinya untuk mendapatkan akses masuk ke dalam halaman *website,* dari mengisi data *user* yang sesuai dengan persyaratan sistem dan diproses oleh sistem sampai dengan disimpan ke *database* kemudian dapat menampilkan halaman *login* untuk *user*. Tampilan dari diagram *sequence register* dapat di lihat pada gambar 4.3.

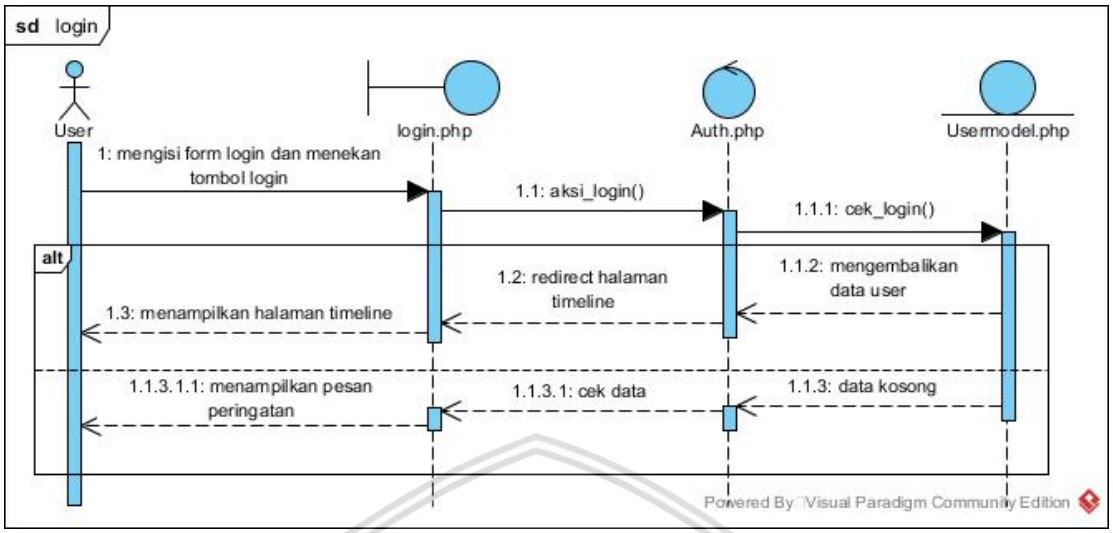


Gambar 4.3 Diagram *Sequence Register*

4.2.3.2 Diagram *Sequence* Login

Alur diagram *sequence* ini menjelaskan alur proses *login* dari *website aplikasi* dalam melakukan validasi data user sekaligus menjadi proses awal dari membentuk sebuah *session* untuk *user*, dari mengisi data *user* hingga *client* dapat melakukan *login* dan diproses oleh sistem sampai dengan sukses kemudian mengakses halaman *timeline*. Dan menampilkan juga kondisi yang lain jika data masukan oleh *user* tidak cocok atau tidak ada pada *database* maka akan menampilkan pesan peringatan bahwa data masukan salah. Proses *login* ini merupakan bagian yang penting karena asal mula sebuah *session* terjadi dan dapat disimpan ke *database*. Tampilan dari diagram *sequence login* dapat di lihat pada gambar 4.4.

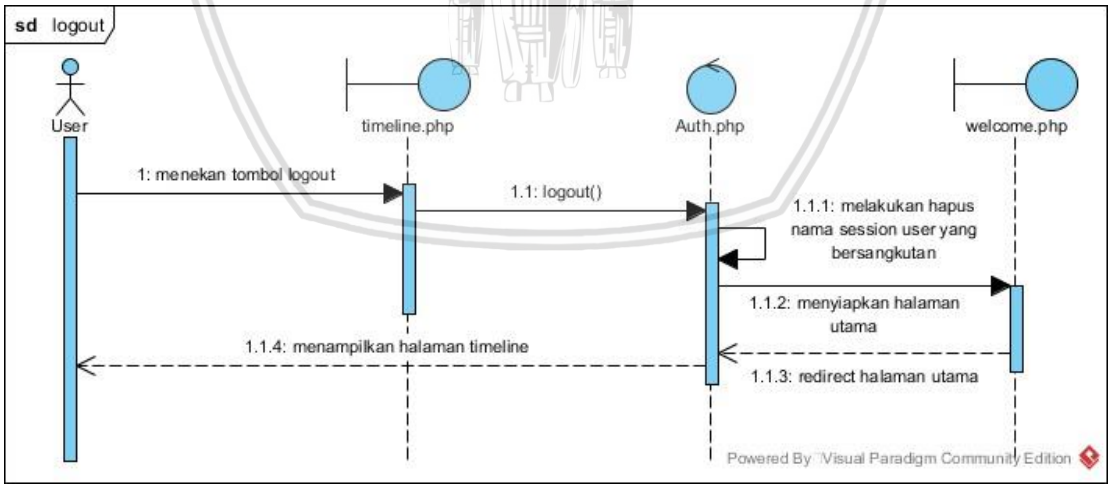
.



Gambar 4.4 Diagram *Sequence Login*

4.2.3.3 Diagram *Sequence* Logout

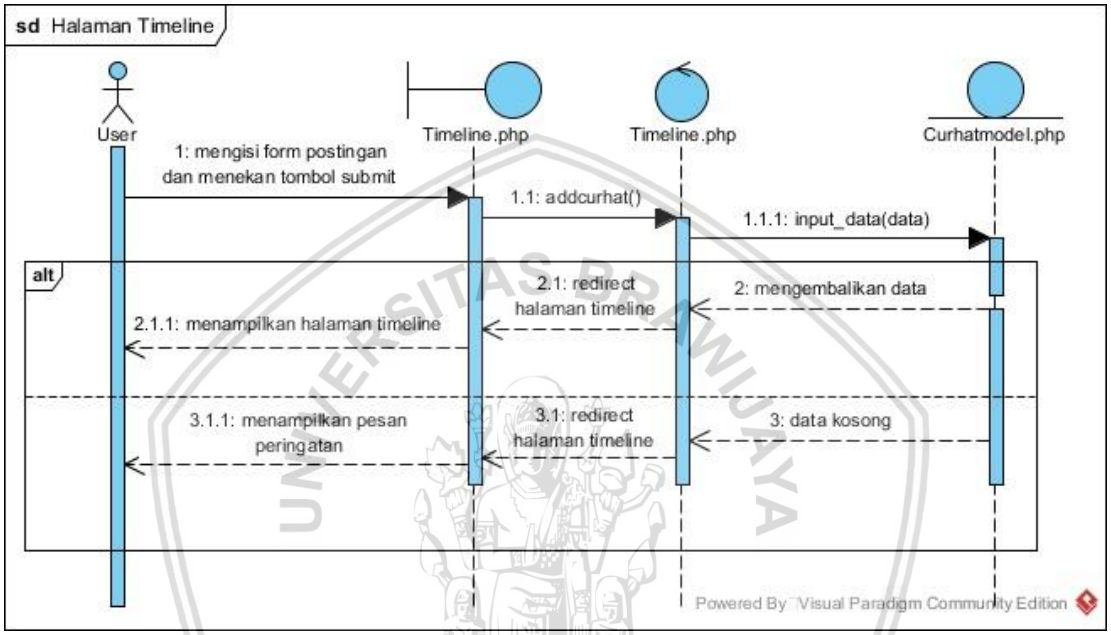
Alur diagram *sequence* ini menjelaskan alur proses *logout* dari *user* pada aplikasi *website* dalam menghapus *session* yang sebelumnya telah dibuat dan disimpan untuk *user*, dari *user* menekan tombol *logout* sehingga sistem akan melakukan hapus data *session* yang sesuai pada *database* sampai dengan sukses kemudian dialihkan untuk mengakses kembali halaman utama aplikasi *website*. Tampilan dari diagram *sequence logout* dapat di lihat pada gambar 4.5.



Gambar 4.5 Diagram *Sequence Logout*

4.2.3.4 Diagram *Sequence* Halaman Timeline

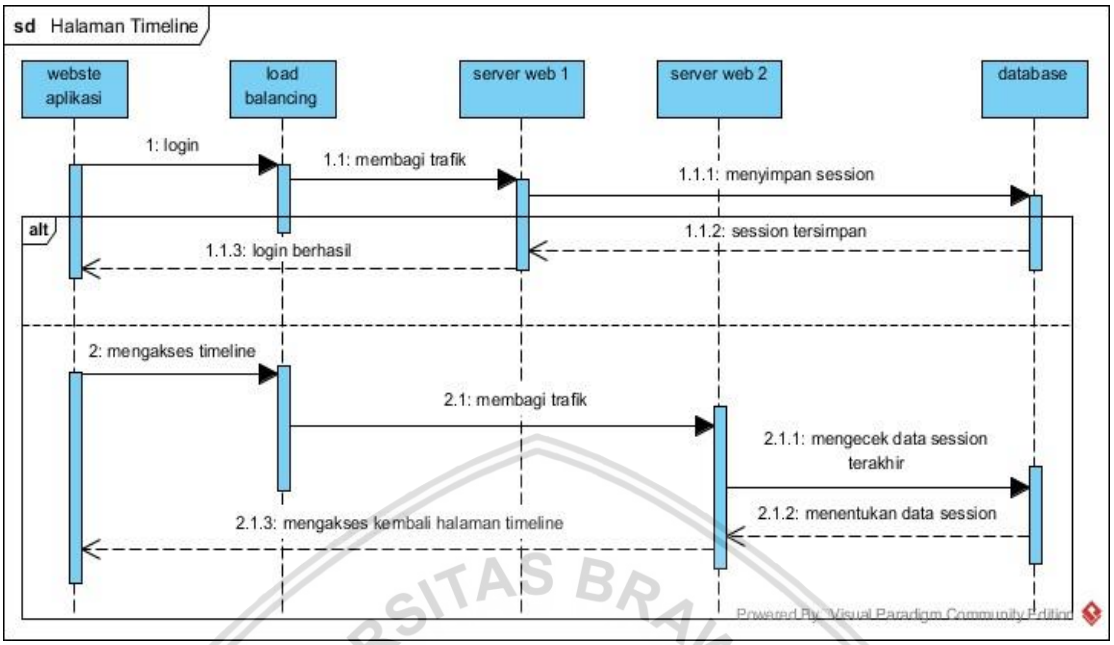
Alur diagram *sequence* ini menjelaskan alur setelah proses *login* dari aplikasi *website* dapat dilakukan sampai dengan sukses kemudian mengakses halaman *timeline*. Dalam halaman *timeline* terdapat fitur berupa menambah postingan. Proses pada halaman *timeline* ini menjelaskan dari fitur menambahkan postingan yang dapat digunakan oleh *user* setelah memasuki halaman *timeline*. Tampilan dari diagram *sequence* halaman *timeline* dapat di lihat pada gambar 4.6.



Gambar 4.6 Diagram *Sequence* halaman *Timeline*

4.2.3.5 Diagram *Sequence* Sistem *Shared Session*

Peneliti melalui *sequence* diagram ingin menjabarkan keseluruhan alur kerja *shared session* pada aplikasi *website* dalam klaster *server web* yang aktif beserta penyimpanan *database* yang bekerja pada aplikasi *website* setelah berhasil melakukan *login* lalu menampilkan halaman *timeline*. Berikut tampilan diagram *sequence* dapat dilihat pada gambar 4.7.



Gambar 4.7 Diagram *Sequence Shared Session*

Alur diagram pada gambar 4.7 yakni menjelaskan sistematika dari aplikasi *website* dalam membangun koneksi *session* pada klaster *server web* di satu *domain web* yang sama. Kemudian *session* disimpan pada *database* hingga *client* melakukan *login* sampai dengan sukses kemudian mengakses halaman *timeline*. Langkah awal yakni dimulai dari mengakses website dengan client *login* menggunakan *username* dan *password* yang telah didaftarkan sebelumnya, kemudian *client* melakukan *login* kedalam *server web* 1 jika berhasil *login* maka *username* dan *password* dari *user* tersebut akan disimpan dalam *database*. Lalu *database* secara otomatis menambahkan data *session* terbaru pada *database* tabel 'ci_*session*' yang dibuat setelah *login* berhasil. Jadi ketika *login* yang dilakukan oleh *client* tersebut sukses dan mampu menampilkan halaman *timeline* pada website aplikasi.

Dalam kondisi selanjutnya pada *sequence* diagram diatas terdapat aktifitas dimana algoritma *Round Robin* berjalan untuk mendistribukan dan membagi trafik ke *web server* 2. Dikarenakan penggunaan *shared session* (tetap pada *domain web* yang sama) pada website ini, maka *server web* 2 akan menyediakan *session ID* baru untuk *user* yang masih berstatus 'login' atau aktif lalu meminta data user terakhir untuk disimpan ke *session ID* baru dari *database* yang akan ditampilkan pada *server web* 2. Untuk itu *user* tidak perlu melakukan autentikasi *login* yang sama seperti sebelumnya. Dengan pengaturan sistem bahwa *website* tersebut akan memperbaharui *session ID* setiap 300 detik maka data *session* yang telah disimpan pada database yang digunakan sebagai autentikasi *login* untuk mengakses *website* selanjutnya, akan diperbarui sesuai waktu yang ditentukan dengan menggunakan *session ID* baru yang disediakan oleh *server web* aktif untuk menangani *request session* yang masuk. Sistem *shared session* bertujuan agar mempermudah kinerja klaster *server web* serta

database dalam menyediakan dan memproses seluruh permintaan *session* dari *client*. Mekanisme ini juga terjadi pada *server web* 3 dan *server web* 4 karena diasumsikan alur proses yang sama persis seperti *server web* 2, jika *server web* 3 dan *server web* 4 bukan dari awal yang menyediakan *session* saat autentikasi *login* untuk *user.*

4.3 Perancangan Pengujian

Pada tahap ini dilakukan pengujian untuk memastikan bahwa sistem berjalan sesuai dengan tujuan yang telah ditentukan dalam penelitian ini. Pengujian sistem tersebut akan dilakukan bedasarkan 2 skema pengujian yang telah direncanakan sebelumnya. Berikut penjabaran 2 pengujian tersebut dijelaskan pada beberapa sub bab.

4.3.1 Pengujian *Login* dan *Logout* menggunakan *Shared Session*

Skema pengujian ini dilakukan untuk mengetahui kinerja sistem *shared session* dari saat awal memulai mengakses *website* dan *user* masuk ke dalam aplikasi *website* atau melakukan *login* hingga melakukan *logout*. Dengan tujuan apakah disaat salah satu web server dinonaktifkan dan disaat yang sama sistem *shared session* mampu bekerja sesuai tugas utamanya. Pengujian kinerja sistem dilakukan dengan menggunakan 1 user virtual yang dikonfigurasikan sedemikian rupa melalui aplikasi uji *Jmeter*. Dalam pengujian ini dibuat *users virtual* sebanyak 1 user virtual yang berbeda dikondisikan melakukan *login* kedalam website dan aplikasi *load balancing* akan mengarahkan ke salah satu *server web* agar menampilkan *website* yang diminta. Namun setelah *login* berhasil *server web* yang sudah diakses dari arahan *load balancing* tersebut mengalami *server down* atau gagal menyediakan layanan untuk user. Karena *load balancing* menggunakan algoritma *Round Robin* maka terjadi mekanisme perpindahan satu *server web* yang mati menuju *server web* yang aktif. Disaat itu juga *server web* yang aktif tidak mempunyai akses *session* yang sama, maka harus meminta akses tersebut ke *database* yang telah menyimpan candangan data *session* pada *database*.

Serta *user* juga melakukan tindakan *logout* agar mengetahui seluruh *session* yang disimpan sebelumnya sudah dihapus, memastikan sudah diakhiri *session* nya serta masing-masing *server web* tidak lagi dapat mengakses *session* yang aktif. Jadi bedasarkan rencana pengujian tersebut akan menampilkan berhasil atau tidak nya dalam menngatasi kondisi tersebut lalu ditampilkan melalui bentuk draft laporan yang menampilkan parameter *connection error* atau *connection success*.

4.3.2 Pengujian *Black box* dari sistem *Shared Session*

Pengujian yang kedua dilakukan yakni untuk mengetahui hasil kinerja metode *shared session* bedasarkan kehandalan dari database dalam menangani permintaan *session* yang dibutuhkan, dengan itu maka dapat diketahui seberapa optimal *server web* dapat bekerja dengan metode *shared session*. Sebagai acuan pengujian

dibutuhkan *user virtual* dengan akun yang berjumlah paling sedikit hingga tingkat paling banyak tersebut, saat pengujian dilakukan bedasarkan kondisi yang telah direncanakan dalam klaster *web server* dikondisikan dimana salah satu hingga tiga *server web* tidak stabil bahkan terjadi *server down*, maka dibutuhkan tindakan *load balancing untuk* mengalihkan ke server yang aktif saat itu. Bedasarkan rencana skenario pengujian yang kedua ini, peneliti ingin mengetahui kinerja sistem *shared session* yang dihasilkan melalui hasil grafik dan tabel dengan menampilkan hasil beberapa parameter *response time, connection error* atau *connection success*. Skenario yang akan diujikan dalam penelitian ini dapat dilihat pada tabel 4.1.

Tabel 4.1 Tabel daftar 3 skenario pengujian *black box*

No. Skenario	*User Virtual*	Status *Server Web*	Percobaan	Total halaman yang diakses dalam *website*	Parameter Hasil
1	100	1 *server web* nonaktif	4 Percobaan	1 kali percobaan mengakses 4 Halaman *website*	*Response time, connection error* atau *success*
2	200	2 *server web* nonaktif	4 Percobaan	1 kali percobaan mengakses 4 Halaman *website*	*Response time, connection error* atau *success*
3	300	3 *server web* nonaktif	4 Percobaan	1 kali percobaan mengakses 4 Halaman *website*	*Response time, connection error* atau *success*

BAB 5 IMPLEMENTASI SISTEM

Pada bab ini adalah hasil dalam mengimplementasikan dari perancangan sistem *shared session* yang akan dibangun pada klaster *server web* melalui *PHP CodeIgniter* dan *database MySQL.* Dan sesuai alur dalam topologi perancangan mulai dari tingkat *backend* hingga *frontend* pada *load balancing* dalam *virtual machine* dengan berpedoman pada metodologi penelitian yang sebelumnya telah dibahas.

5.1 Implementasi Perangkat Pendukung

Berikut ini adalah tahapan awal yang dibagi dalam beberapa aplikasi pendukung untuk membagun seluruh *node* klaster *server web* melalui *Linux Ubuntu Server* yang berjumlah 5 *node* dengan fungsi yang berbeda-beda dan dipasangkan pada masing-masing tiap mesin *virtual* melalui aplikasi *VirtualBox*.

5.1.1 Implementasi *Load Balancing*

Pada implementasi *load balancing* ini perangkat lunak yang diterapkan adalah *NGINX* yang dipasangkan di salah satu *node server*. *Node server* tersebut memiliki alamat IP 192.168.56.101. *Load balancing* ini memiliki peran sebagai perangkat yang mendistribusikan trafik data secara seimbang masuk ke seluruh *node server* pada klaster *server web* sehingga dapat menigkatkan kinerja *server web* saat itu. Dalam mengkofigurasi *NGINX* agar dapat melakukan penyeimbangan beban muatan digunakan algoritma *Round Robin* secara bawaan setelah memasang perangkat *NGINX*.

5.1.1.1 Konfigurasi *NGINX*

Langkah awal sebelum konfigurasi aplikasi *NGINX* yakni menginstal terlebih dahulu aplikasi *NGINX* tersebut yang sudah memiliki algoritma bawaan *Round Robin*. Peneliti menginstal *NGINX* dengan perintah yang diberikan di bawah ini:

```
sudo apt-get install nginx
```

Setelah aplikasi selesai di instal, kemudian melakukan perubahan pada file konfigurasi yang bertujuan untuk mengaktifkan *load balancing* dengan *upstream.* File konfigurasi *default* bernama *loadbalancer.conf* dari *NGINX* ada di sistem *Ubuntu* dan terletak di */etc/nginx/conf.d.* Berikut perubahan konfigurasi yang digunakan:

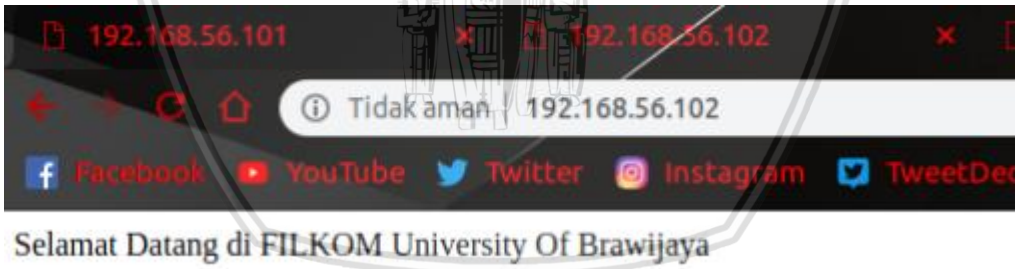
```
upstream backend {
        server 192.168.56.102;
        server 192.168.56.103;
        server 192.168.56.104;
        server 192.168.56.105;
```

```
}
server {
        listen 80;
          location / {
          proxy_pass http://backend;
        }
}
```

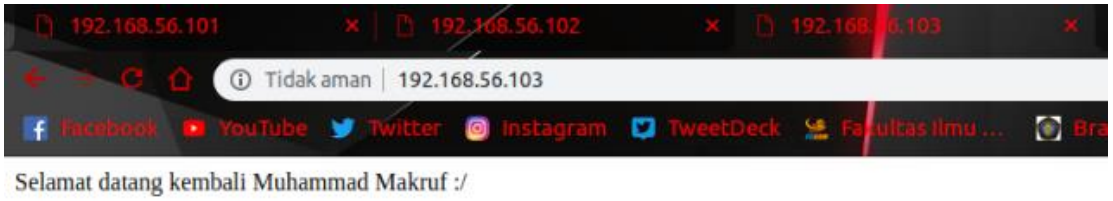
Perubahan konfigurasi perlu dilakukan agar membantu aplikasi *NGINX* untuk *listen* nama *Domain* atau *IP Publik*, setelah seluruhnya sudah selesai dilakukan aplikai perlu dijalankan ulang. Berikut perubahan konfigurasi yang digunakan:

```
sudo service nginx restart
```

Setelah proses menghidupkan ulang *service NGINX* selesai, lalu menguji pada *web server* dapat dijalankan hanya dengan membuka nama *host* atau *IP load balancing* di *browser*. Untuk memastikan bahwa mekanisme sudah *Round Robin* berjalan, jadi peneliti memiliki 4 *node server NGINX default* dan berada di belakang *load balancing*, dengan kondisi masing-masing dari 4 *server* tersebut memiliki halaman *indeks* yang sudah dimodifikasi dan *server Apache* sudah selesai diinstal. Berikut tampilan halaman yang sudah dimodifikasi dapat dilihat pada gambar 5.1 sampai dengan gambar 5.4.

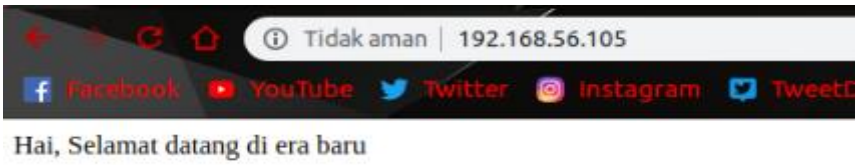


Gambar 5.1 Tampilan halaman *indeks* pada *server* IP 192.168.56.102



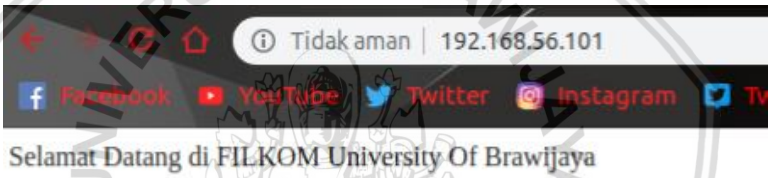
Gambar 5.2 Tampilan halaman *indeks* pada *server* IP 192.168.56.103

Gambar 5.3 Tampilan halaman *indeks* pada *server* IP 192.168.56.104

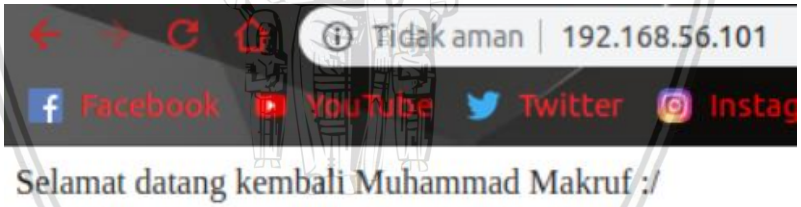


Gambar 5.4 Tampilan halaman *indeks* pada *server* IP 192.168.56.105

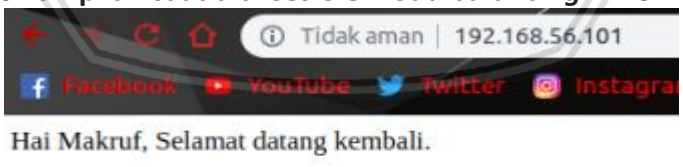
Berikut mekanisme *Round Robin* yang ditampilkan melalui halaman indeks yang sudah diubah dan diakses melalui *server load balancing* IP 192.168.56.101 dapat dilihat pada gambar 5.5 sampai dengan gambar 5.8.



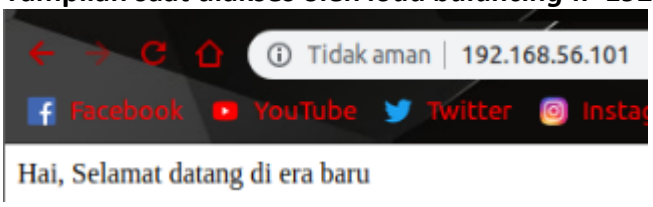
Gambar 5.5 Tampilan saat diakses oleh *load balancing* IP 192.168.56.101



Gambar 5.6 Tampilan saat diakses oleh *load balancing* IP 192.168.56.101



Gambar 5.7 Tampilan saat diakses oleh *load balancing* IP 192.168.56.101



Gambar 5.8 Tampilan saat diakses oleh *load balancing* IP 192.168.56.101

5.1.2 Implementasi *Web Server*

Pada implementasi *web server* ini perangkat lunak yang diterapkan adalah *Apache* yang dipasang pada 4 *node server*. Node server tersebut memiliki alamat IP 192.168.56.102, IP 192.168.56.103, IP 192.168.56.104 dan IP 192.168.56.105. *Apache* ini memiliki peran sebagai yang bertanggung jawab untuk menangani *request-response HTTP* serta *logging* informasi menggunakan basis data, secara detail yang akan diinstal pada masing-masing *server web* dalam satu klaster *server web*.

5.1.2.1 Konfigurasi *Apache*

Langkah awal sebelum konfigurasi aplikasi *Apache* yakni menginstal terlebih dahulu aplikasi *Apache* tersebut pada masing-masing *node server*. Peneliti menginstal Apache dengan perintah yang diberikan di bawah ini:

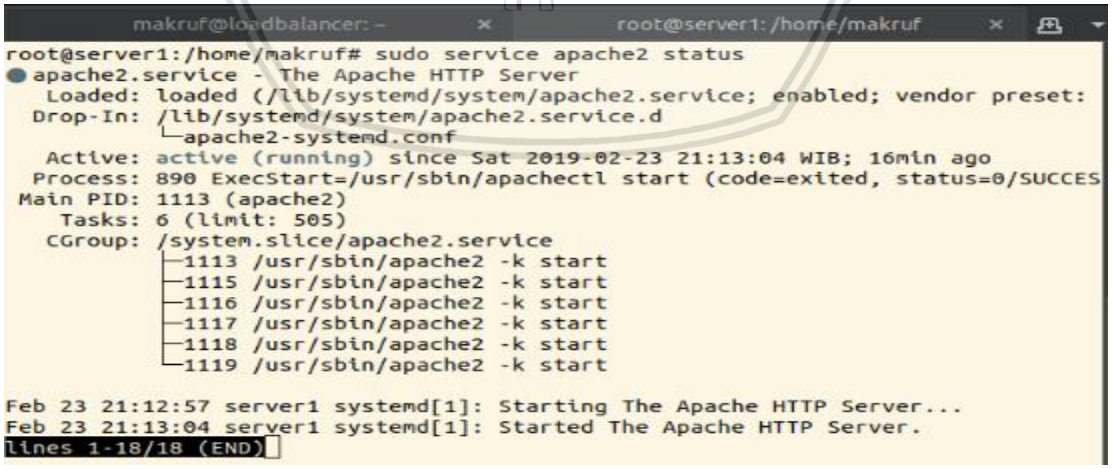
```
sudo apt-get install apache2
```

Sebelum menguji *Apache*, peneliti memeriksa pengaturan *firewall* untuk akses luar ke port web *default*. Tujuan *firewall UFW* dikonfigurasikan yakni untuk membatasi akses ke *server*. Namun selama instalasi, *firewal UFW* tidak digunakan pada penelitian ini jadi diasumsikan pengaturan mengikuti bawaan aplikasi yakni di nonaktifkan akses ke *Apache* pada *firewall*.

Proses instalasi dan perubahan konfigurasi dari *Apache* hampir selesai. *Server web* seharusnya sudah aktif dan berjalan. Untuk memastikan bahwa perubahan sudah benar periksa status layanan sistem dengan perintah yang diberikan di bawah ini:

```
sudo systemctl status apache2
```

Kemudian dari perintah tersebut akan menampilkan status layanan *Apache* dari salah satu *server web* saat itu yang dapat dilihat pada gambar 5.9.

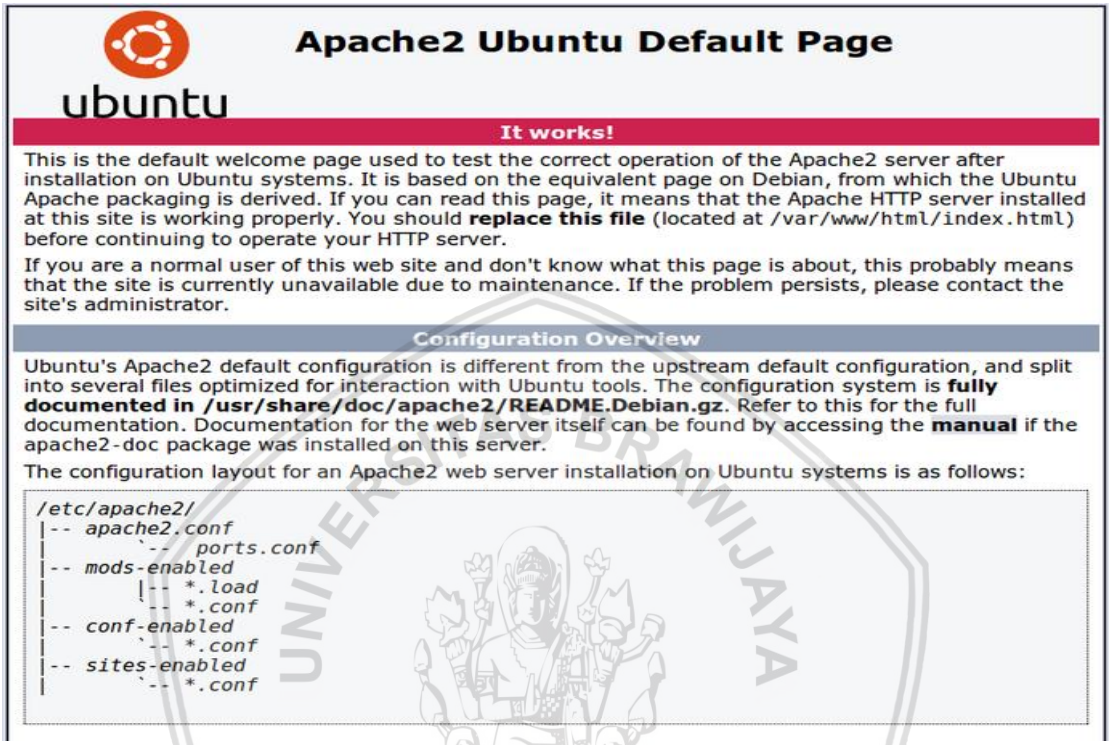


Gambar 5.9 Tampilan status layanan *Apache* dan sudah aktif

Untuk memastikan bahwa memasukkan alamat IP *server* 192.168.56.102, dimasukkan ke bilah alamat *browser* dengan perintah yang diberikan di bawah ini:

```
http://192.168.56.102
```

Kemudian dari perintah tersebut akan menampilkan halaman *web default Apache* yang dapat dilihat pada gambar 5.10.



Gambar 5.10 Tampilan halaman *web default Apache*

5.1.2.2 Menyiapkan *Host Virtual Apache*

Setelah instalasi dan konfigurasi aplikasi *Apache* seluruhanya selesai, peneliti menyiapkan folder dokumen khusus pada direktori */var/www/html*. Folder dari struktur direktori tersebut merupakan bawaan dari aplikasi *Apache*. Peneliti disini menambahkan direktori baru untuk menempatkan kode program agar bisa diakses pada aplikasi *Apache* dengan perintah yang diberikan di bawah ini:

```
sudo mkdir -p /var/www/html/skripsi
```

Menambahkan izin akses ke seluruh direktori pada *root web* agar bisa diakses pada aplikasi *Apache* dengan perintah yang diberikan di bawah ini:

```
sudo chmod -R 755 /var/www/html/skripsi
```

Kemudian menambahkan izin akses ke seluruh direktori pada *root web* agar bisa diakses pada aplikasi *Apache* dengan perintah yang diberikan di bawah ini:

```
sudo chmod -R 755 /var/www/html/skripsi
```

Selanjutnya melakukan perubahan pada file konfigurasi yang bertujuan untuk mengizinkan seluruh file di dalam direktori */var/www/html/skripsi* dapat diakses lebih

lanjut. File konfigurasi global *apache2.conf* ada di sistem *Ubuntu* dan terletak di */etc/apache2.* Berikut perubahan konfigurasi yang digunakan:

```
<Directory />
  Options FollowSymLinks
  AllowOverride None
  Require all denied
</Directory>

<Directory /usr/share>
   AllowOverride None
   Require all granted
</Directory>

<Directory /var/www/>
   Options Indexes FollowSymLinks
   AllowOverride None
   Require all granted
</Directory>

<Directory /var/www/html>
   AllowOverride All
</Directory>
```

Tahap instalasi dan konfigurasi aplikasi *Apache* telah selesai, selanjutnya instalasi dapat dilakukan pada *node server* yang lain dengan IP 192.168.56.103, 192.168.56.104, dan 192.168.56.105 dikarenakan alur intalasi kebutuhan aplikasi yang pada dasarnya sama seperti penjelasan dari *node server* IP 192.168.56.102 yang telah dijabarkan oleh peneliti pada sub bab ini.

5.1.3 Implementasi *Database MySQL*

Pada implementasi ini dikondisikan klaster *web server* ini sudah berjalan dan perangkat lunak selanjutnya yang diterapkan adalah *Database MySQL*. Perangkat lunak tersebut berada pada *node server* dengan alamat IP 192.168.56.101 dan berada pada *node server* yang sama seperti IP *server load balancin*g. *Database MySQL* ini memiliki peran sebagai sistem manajemen basis data. Pada dasarnya, yang mengatur dan menyediakan akses ke *database* sekaligus dapat menyimpan informasi.

5.1.3.1 Instalasi *MySQL*

Langkah awal yakni menginstal terlebih dahulu aplikasi *MySQL* tersebut pada node server IP 192.168.56.101. Peneliti menginstal MySQL dengan perintah yang diberikan di bawah ini:

```
sudo apt install mysql-server
```

Ketika instalasi selesai selanjutnya menjalankan skrip keamanan yang diperlukan pada *MySQL* untuk mengubah pengaturan akses bawaan dari sistem. Skrip interaktif berupa perintah yang diberikan di bawah ini:

```
sudo mysql_secure_installation
```

Masukan dari perintah diatas akan menjelaskan untuk penggunaan kata sandi saat masuk ke dalam *root* sistem *database MySQL*. Perintah sebelumnya akan menampilkan pertanyaan valiadasi seperti di bawah ini:

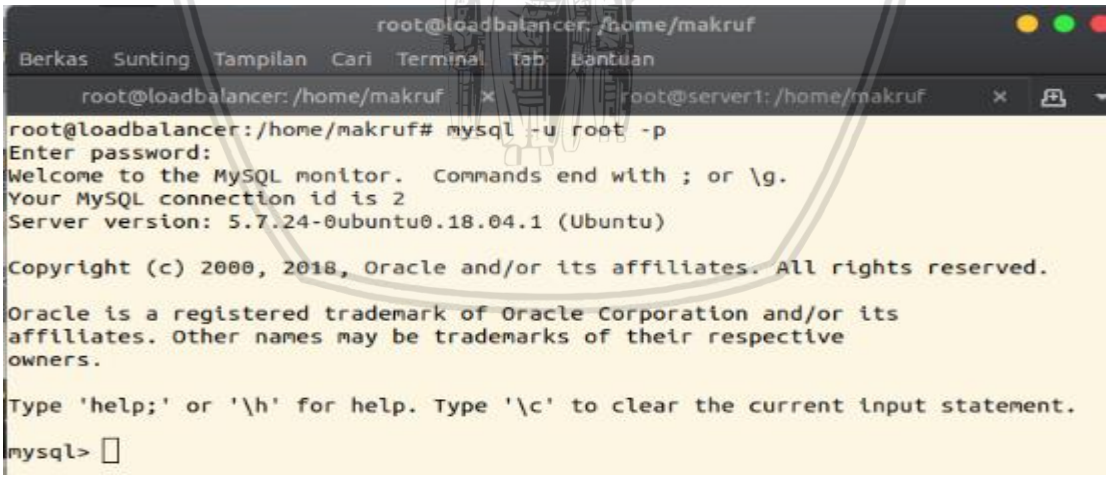
```
VALIDATE PASSWORD PLUGIN can be used to test passwords and
improve security. It checks the strength of password and allows
the users to set only those passwords which are secure enough.
Would you like to setup VALIDATE PASSWORD plugin?

Press y|Y for Yes, any other key for No:
```

Dalam penelitian ini peneliti memutuskan untuk tidak memakai kata sandi untuk masuk ke sistem *root* pada *MySQL* dikarenakan untuk mempermudah kebutuhan penelitian. Selanjutnya setelah pengaturan diatas selesai bisa mengakses *database* dengan perintah seperti di bawah ini:

```
sudo mysql –u root –p
```

Kemudian dari perintah tersebut akan menampilkan masuk ke dalam sistem *database* yang dapat dilihat pada gambar 5.11.



Gambar 5.11 Tampilan saat masuk sistem *root* pada *MySQL*

Selanjutnya memberikan pengaturan hak akses pada *database* agar setiap *user* di setiap masing-masing *node* klaster *server web* diberi izin akses dengan *query* yang diperbolehkan, seperti: **SELECT, INSERT, UPDATE, DELETE**, dan lainnya. Format *query* tersebut seperti di bawah ini:

```
mysql> CREATE USER ‘makruf’@’192.168.56.102’ IDENTIFIED BY
‘makruf’;
```

```
mysql> GRANT ALL PRIVILEGES ON *.* TO
'makruf'@'192.168.56.102' WITH GRANT OPTION;
mysql> CREATE USER 'root'@'192.168.56.102' IDENTIFIED BY
'makruf';
mysql> GRANT ALL PRIVILEGES ON *.* TO
'root'@'192.168.56.102' WITH GRANT OPTION;

mysql> CREATE USER 'makruf'@'192.168.56.103' IDENTIFIED BY
'makruf';
mysql> GRANT ALL PRIVILEGES ON *.* TO
'makruf'@'192.168.56.103' WITH GRANT OPTION;
mysql> CREATE USER 'root'@'192.168.56.103' IDENTIFIED BY
'makruf';
mysql> GRANT ALL PRIVILEGES ON *.* TO
'root'@'192.168.56.103' WITH GRANT OPTION;

mysql> CREATE USER 'makruf'@'192.168.56.104' IDENTIFIED BY
'makruf';
mysql> GRANT ALL PRIVILEGES ON *.* TO
'makruf'@'192.168.56.104' WITH GRANT OPTION;
mysql> CREATE USER 'root'@'192.168.56.104' IDENTIFIED BY
'makruf';
mysql> GRANT ALL PRIVILEGES ON *.* TO
'root'@'192.168.56.104' WITH GRANT OPTION;

mysql> CREATE USER 'makruf'@'192.168.56.105' IDENTIFIED BY
'makruf';
mysql> GRANT ALL PRIVILEGES ON *.* TO
'makruf'@'192.168.56.105' WITH GRANT OPTION;
mysql> CREATE USER 'root'@'192.168.56.105' IDENTIFIED BY
'makruf';
mysql> GRANT ALL PRIVILEGES ON *.* TO
'root'@'192.168.56.105' WITH GRANT OPTION;
```

Setelah pengaturan diatas selesai dapat keluar dari sistem *MySQL* dengan perintah seperti di bawah ini:

```
mysql> /q
```

Dalam penelitian ini peneliti sudah menyiapkan file *MySQL* bernama *pesbuk.sql* yang berisikan data tabel untuk mendukung pengujian sistem *shared session* sekaligus

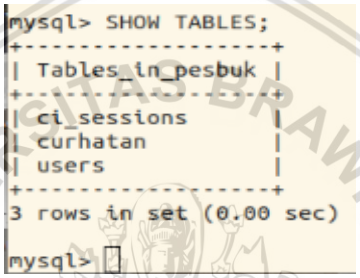
mempercepat dan mempermudah kebutuhan penelitian. Perintah tersebut seperti di bawah ini:

```
mysql –u root –p pesbuk < pesbuk.sql
```

Lalu masuk kembali ke sistem *database MySQL* untuk memeriksa file *pesbuk.sql* sudah dapat diakses atau belum dengan perintah seperti di bawah ini:

```
mysql –u root –p
mysql> connect pesbuk;
mysql> SHOW TABLES;
```

Kemudian menampilkan hasil berupa isi *database* 'pesbuk' dapat dilihat pada gambar 5.12.

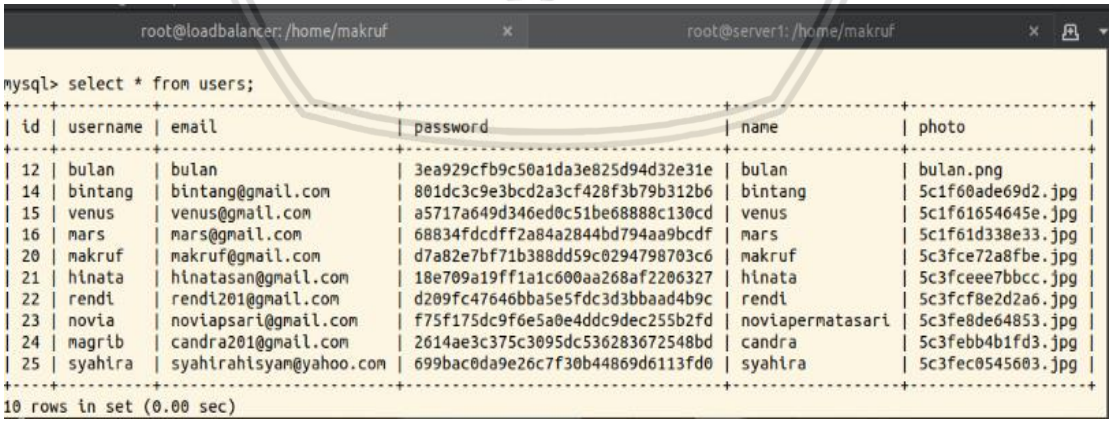


Gambar 5.12 Tampilan daftar tabel pada *database* 'pesbuk' *MySQL*

Lalu masukkan kueri baru di sistem *database MySQL* untuk menampilkan isi dari tabel *user* yakni dengan perintah seperti di bawah ini:

```
mysql> select * from users;
```

Kemudian menampilkan hasil data *user* yang sudah didaftarkan dan tersimpan berupa isi tabel 'users' dan dapat dilihat pada gambar 5.13.



Gambar 5.13 Tampilan data dari tabel 'users' pada database MySQL

```
mysql> select * from ci_sessions;
```

Kemudian menampilkan hasil data *session* terakhir yang tersimpan dari *user*, data tersebut tersaji dalam tabel 'ci_sessions' dan dapat dilihat pada gambar 5.14.

Gambar 5.14 Tampilan data dari tabel 'ci_sessions' pada *database MySQL*

```
mysql> select * from curhatan;
```

Kemudian menampilkan hasil data postingan terakhir yang tersimpan dari *user*, data tersebut tersaji dalam tabel 'curhatan' dan dapat dilihat pada gambar 5.15.

```
mysql> select * from curhatan;
+----+----------+--------------------------------------+
| id | id_uwong | isi                                  |
+----+----------+--------------------------------------+
|  1 |        0 | Sore Gan                             |
|  6 |       12 | makruf                               |
|  8 |       14 | hai makruf                           |
|  9 |       14 | tes                                  |
| 10 |       15 | hai makruf                           |
| 11 |       16 | hai gan                              |
| 12 |       12 | assalamualaikum rek                  |
| 14 |       16 | haiii broo                           |
| 17 |       17 | halooo kawan                         |
| 18 |       23 | halooo juga, aku boleh kenalan kah?  |
| 22 |       23 | hai nov, lgi ngapain samean?         |
| 23 |       23 | emmmm                                |
| 37 |       20 | heyy, assalamualaikum kawan          |
| 40 |       20 | kamuu kenapa?                        |
| 44 |       23 | aku capekk ini :((                   |
+----+----------+--------------------------------------+
15 rows in set (0.00 sec)

mysql>
```

Gambar 5.15 Tampilan data dari tabel 'curhatan' pada *database MySQL*

5.1.4 Implementasi *PHP*

Pada implementasi *web server* ini perangkat lunak yang diterapkan adalah *PHP* yang diinstalasikan pada 4 *node server*. *Node server* tersebut memiliki alamat IP 192.168.56.102, IP 192.168.56.103, IP 192.168.56.104 dan IP 192.168.56.105. *PHP* ini memiliki peran sebagai yang perangkat yang mengatur dan memproses kode program dengan menjalankan *script*, lalu terhubung ke *database MySQL* untuk mendapatkan informasi, dan menyerahkan konten yang diproses ke *server web* setiap *server web* untuk ditampilkan. Perangkat ini akan di instal pada masing-masing klaster *server web*.

5.1.4.1 Instalasi *PHP*

Langkah awal yakni menginstal terlebih dahulu aplikasi *PHP* tersebut pada masing-masing *node server*. Peneliti menginstal *PHP* dengan perintah yang diberikan di bawah ini:

```
sudo apt install php libapache2-mod-php php-mysql
```

Sebelum menggunakan aplikasi *PHP*, mulai ulang server *web Apache* agar perubahan saat setelah instalasi. Dengan perintah yang diberikan di bawah ini:

```
sudo systemctl restart apache2
```

Untuk menguji sistem bawaan aplikasi *PHP* sudah dikonfigurasikan dengan benar. Peneliti membuat skrip *php* yang disebut *info.php*, agar *Apache* dapat menemukan file skrip ini maka harus disimpan pada lokasi yang berada di dalam direktori */var/www/html/.* Dengan perintah membuat file baru seperti di bawah ini:

```
sudo nano /var/www/html/info.php
```

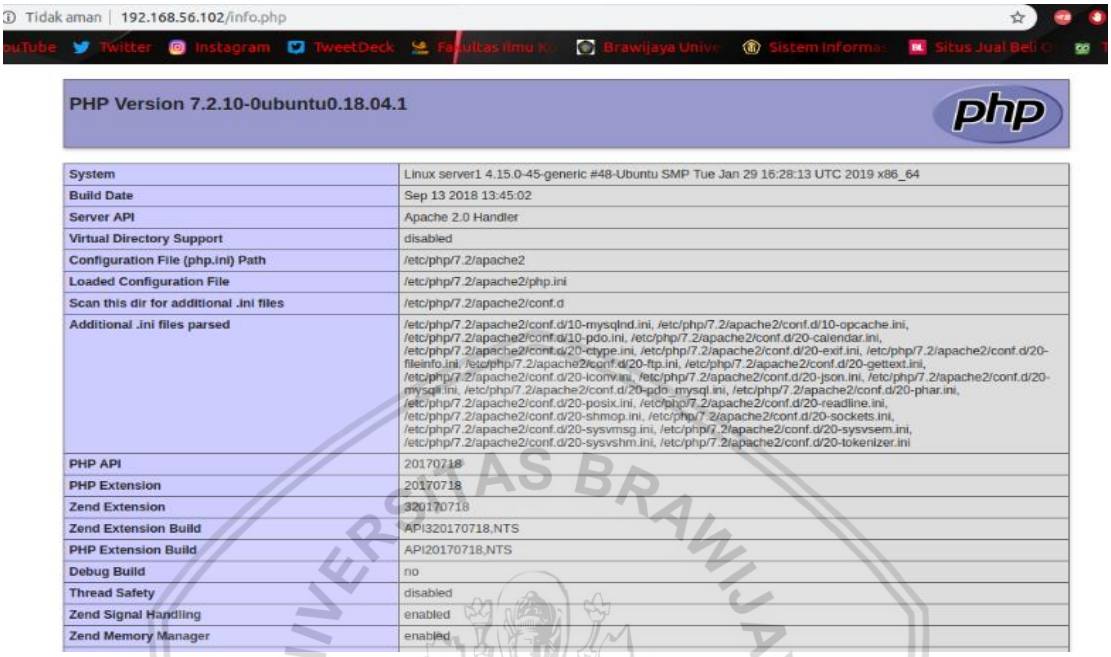
Kemudian akan muncul halaman kosong pada file tersebut dan peneliti membuat kode program seperti yang diberikan di bawah ini:

```
<?php
phpinfo();
?>
```

Lalu simpan dan tutup file tersebut, kemudian buka skrip file dapat di buka melalui *web browser* dengan alamat IP masing-masing *server*. Dengan alamat *URL* yang diberikan di bawah ini:

```
http://192.168.56.102/info.php
```

Kemudian dari perintah tersebut akan menampilkan halaman *web* seperti yang dapat dilihat pada gambar 5.16.



Gambar 5.16 Tampilan halaman *web* informasi *Apache* melalui *PHP*

Tahap instalasi dan konfigurasi aplikasi *PHP* telah selesai, selanjutnya instalasi dapat dilakukan pada *node server* yang lain dengan alamat *IP* 192.168.56.103, 192.168.56.104, dan 192.168.56.105 dikarenakan alur intalasi kebutuhan aplikasi yang pada dasarnya sama seperti penjelasan dari *node server IP* 192.168.56.102 yang telah dijabarkan oleh peneliti pada sub bab ini.

5.2 Implementasi *Shared Session* melalui *PHP CodeIgniter*

Berikut ini adalah kode program implementasi s*hared session* yang digunakan untuk menyelesaikan permasalahan pada penelitian ini dengan mempergunakan *PHP CodeIgniter* beserta *library* yang ada. Berikut ini adalah kode implementasi yang digunakan pada *PHP CodeIgniter* untuk menampilkan aplikasi *website* yang sudah dirancang dan diterapkan pada masing-masing *server web*.

5.2.1 *Class* Auth.php

Pada potongan kode program dibawah ini adalah kode dari kelas *controller* yang dibutuhkan untuk membangun program *shared session* pada tabel 5.1.

Tabel 5.1 *Source-code Class* Auth.php

```
1  <?php
2  class Auth extends CI_Controller{
3    function __construct(){
4        parent::__construct();
```

```
5         $this->load->model('usermodel');
      }
```

Penjelasan kode program:

1-2 Potongan kode digunakan untuk mendeklarasikan *class* yang bernama *Auth* yang menge-*extend class* inti *Codeigniter.*

3-5 Potongan kode digunakan untuk mendeklarasikan fungsi dalam class lalu akan menjalankan class nya ketika di inisialisasi kemudian mengakses class pada 'usermodel'.

```
6   function aksi_login(){
7               $username = $this->input->post('username');
8               $password = $this->input->post('password');
9               $where = array(
10                      'username' => $username,
11                      'password' => md5($password)
                        );
12              $cek=$this->usermodel->cek_login("users",$where)-
    >num_rows();
13        $datauser=$this->usermodel->cek_login("users",$where)-
    >result();
```

Penjelasan kode program:

6-8 Potongan kode digunakan untuk mendeklarasikan fungsi login lalu akan menginisiasi parameter dengan masukan berupa username dan password.

9-11 Potongan kode digunakan untuk memberi nilai parameter masukan dimana username berupa string dan password berupa nilai hash md5.

12-13 Potongan kode digunakan untuk mengecek melalui fungsi login pada class usermodel dengan melihat pada tabel '*users*' sesuai baris database user dan jika berhasil menampilkan data user sesuai masukan yang diberikan.

```
14        if($cek > 0){
15              $data_session = array(
16                      'nama' => $username,
17                      'status' => "login",
18                      'datauser'=>$datauser,
           );
```

Penjelasan kode program:

14-18 Potongan kode digunakan untuk mengecek jika data *user* ada maka akan membuat kueri baru dari data tersebut ke tabel '*session*' beserta data *user* secara keseluruhan.

```
19                      $this->session->set_userdata($data_session);
20                      redirect(base_url("pages/timeline"));

21              }else{
22                      echo "Username dan password salah !";
                }
        }
```

Penjelasan kode program:

19-20 Potongan kode digunakan untuk mengeset data session dan akan disimpan dalam *server* lalu melakukan masuk ke halaman timeline.

21-22 Potongan kode digunakan untuk memberikan pesan kesalahan masukan pada *user* jika *username* dan *password* tidak sesuai dan tidak ada pada *database*.

```
23    public function register(){
24                $nama = $this->input->post('name');
25                $username = $this->input->post('username');
26                $email = $this->input->post('email');
27                $password = $this->input->post('password');
28                $password = md5($password);
29                $gambarfoto=uniqid();
30                $config['upload_path']          = './assets/image/profil/';
31                $config['allowed_types']        = 'gif|jpg|png';
32                $config['max_size']             = 5000;
33                $config['max_width']            = 3000;
34                $config['max_height']           = 3000;
35                $config['file_name']            = $gambarfoto;
```

Penjelasan kode program:

23-28 Potongan kode digunakan untuk mendeklarasikan fungsi register lalu akan menginisiasi parameter dengan masukan berupa nama, email, foto profil, username dan password.

29-35 Potongan kode digunakan untuk memberikan kofigurasi pada data gambar sesuai ketentuan yang diperbolehkan untuk user.

```
36          $this->load->library('upload', $config);
37                if ( ! $this->upload->do_upload('photo')){
38                      $data['error']=$this->upload->display_errors();
39                      $this->load->view('register', $data);

40          }else{
41                      $upload_data = $this->upload->data();
42                      $file_name = $upload_data['file_name'];
43                      $data = array(
44                            'name' => $nama,
45                            'username' => $username,
46                            'email' => $email,
47                            'password' => $password,
48                            'photo' => $file_name,
                              );
49                      $this->usermodel->input_data($data);
50                      redirect(base_url("pages/login"));
                }
          }
```

Penjelasan kode program:

36-39 Potongan kode digunakan untuk mengunggah data foto user jika data foto tidak sesuai ketentuan yang diberikan maka akan menampilkan pesan ‘erorr’ dan kembali mendaftarkan data ulang.

40-50 Potongan kode digunakan untuk mengunggah data user jika data sesuai ketentuan yang diberikan maka akan dimasukan ke database tabel ‘user’ dan dialihkan ke halaman login untuk memasukkan username dan password.

```
51          function logout(){
52                  $this->session->sess_destroy();
53                  redirect(base_url("pages"));
            }
    }
    ?>
```

Penjelasan kode program:

52-54 Potongan kode digunakan untuk mendeklarasikan fungsi logout lalu akan mengecek pada data session yang sesuai untuk menghapus session user tersebut dan dialihkan ke halaman home pada website

5.2.2 *Class* Timeline.php

Pada potongan kode program dibawah ini adalah kode dari kelas *controller* yang dibutuhkan untuk membangun program *shared session* pada tabel 5.2.

Tabel 5.2 *Source-code Class* Timeline.php

```
1   <?php
2   defined('BASEPATH') OR exit('No direct script access allowed');
3   class Timeline extends CI_Controller {
4           function __construct(){
5               parent::__construct();
6               $this->load->model('curhatmodel');
            }
```

Penjelasan kode program:

1-2 Potongan kode digunakan untuk sintaks *PHP* yang memastikan tidak ada akses script secara langsung, dan harus diakses melalui alamat website yang sudah dikonfigurasikan.

3-6 Potongan kode digunakan untuk class yang bernama Timeline yang menge-extend class inti *Codeigniter* sekaligus mendeklarasikan fungsi dalam class lalu akan menjalankan *class* nya ketika di inisialisasi kemudian mengakses class pada 'curhatmodel'.

```
7       public function addcurhat(){
8           $id = $this->input->post('iduser');
9           $isicurhat = $this->input->post('isicurhat');
10          $data = array(
11            'id_uwong' => $id,
12            'isi' => $isicurhat,
            );
```

Penjelasan kode program:

7-12 Potongan kode digunakan untuk membuat postingan baru yang dideklarasikan melalui fungsi addcurhat, lalu akan menginisiasi parameter dengan masukan berupa iduser dan isicurhat. Dan membuat kueri data baru pada tabel 'curhatan' sesuai kolom id dan isicurhat tersebut.

```
13          $this->curhatmodel->input_data($data);
14          redirect('pages/timeline');
        }
```

```
15      public function hapus_curhat($id){
16        $where = array('id' => $id);
17        $this->curhatmodel->delete_data($where);
18        redirect('pages/timeline');
        }
    }
```

Penjelasan kode program:

13-14 Potongan kode digunakan untuk melakukan memasukan data ke database melalui fungsi pada class 'curhatmodel' dan jika berhasil maka dialihkan ke halaman timeline.

15-18 Potongan kode digunakan untuk menghapus postingan yang dideklarasikan melalui fungsi hapus_curhat, lalu akan menginisiasi parameter yang sesuai dengan id pada tabel 'curhatan'. Jika ditemukan kueri data yang akan dihapus pada tabel 'curhatan' sesuai kolom id dan isicurhat tersebut maka data dihapus kemudian dialihkan ke halaman timeline.

5.2.3 *Class* Pages.php

Pada potongan kode program dibawah ini adalah kode dari kelas *controller* yang dibutuhkan untuk membangun program *shared session* pada tabel 5.3.

Tabel 5.3 *Source-code Class* Pages.php

```
1   <?php
2   defined('BASEPATH') OR exit('No direct script access allowed');
3   class Pages extends CI_Controller {
4           function __construct(){
5               parent::__construct();
6               $this->load->model('usermodel');
7               $this->load->model('curhatmodel');
            }
```

Penjelasan kode program:

1-2 Potongan kode digunakan untuk sintaks PHP yang memastikan tidak ada akses script secara langsung, dan harus diakses melalui alamat website yang sudah dikonfigurasikan.

3-7 Potongan kode digunakan untuk class yang bernama Pages yang menge-extend class inti Codeigniter sekaligus mendeklarasikan fungsi dalam class lalu akan menjalankan class nya ketika di inisialisasi kemudian mengakses class dari 'usermodel' dan 'curhatmodel'.

```
8           public function index(){
9                   $this->load->view('home');
            }
```

Penjelasan kode program:

8-9 Potongan kode digunakan untuk mendeklarasikan fungsi halaman index yang akan diinisiasi oleh class 'home' yang terdapat pada folder 'view'.

```
10  public function login(){
11              if ($this->session->userdata("status")=="login"){
12                      $where = array(
13                      'username'=>$this->session->userdata("nama")
                        );
14          $datauser=$this->usermodel->cek_login("users",$where)->row();
```

Penjelasan kode program:

10-14 Potongan kode digunakan untuk menjalankan fungsi login, mengecek jika user telah login atau belum maka akan mencari id dan username pada tabel 'users' sesuai dengan masukan data user saat login.

```
15  $data['user']=$datauser;
16          if($datauser){
17             $dat['nama'] = $datauser->username;
18              $dat['iduser'] = $datauser->id;
```

Penjelasan kode program:

15-18 Potongan kode digunakan untuk membuat variabel 'user' bedasarkan dari parameter datauser, lalu akan menginisiasi parameter dengan mengecek dari username dan id pada datauser.

```
20  $wherecurhat = array('id_uwong' => $dat['iduser']);
21  $result=$this->curhatmodel->get_data("curhatan",$wherecurhat)-
    >result();
22                              $data['curhat']=$result;
23                              $this->load->view('timeline',$data);
                        }
                }
```

Penjelasan kode program:

20-21 Potongan kode digunakan untuk mencari data curhatan bedasarkan 'id_uwong' dan data pada iduser.

22-23 Potongan kode digunakan untuk menampilkan seluruh hasil data curhatan user yang telah login dan ditampilkan keseluruhan pada halaman timeline.

```
24  else{
25          $this->load->view('login');
                    }
            }
26  public function register(){
27          $this->load->view('register');
     }
```

Penjelasan kode program:

24-25 Potongan kode digunakan untuk jika tidak data yang dimasukan kembali menampilkan halaman login.

26-27 Potongan kode digunakan untuk menjalankan fungsi register yang akan di eksekusi oleh class register yang terdapat pada folder view.

```
28          public function timeline(){
29                  if ($this->session->userdata("status")=="login"){
30                          $where = array(
31                          'username'=> $this->session->userdata("nama")
```

```
                        );
32                      $datauser=$this->usermodel-
    >cek_login("users",$where)->row();
33                      $data['user']=$datauser;
34                      if($datauser){
35                              $dat['nama'] = $datauser->username;
36                               $dat['iduser'] = $datauser->id;

37                              $wherecurhat = array(
38                                      'id_uwong' => $dat['iduser']
                                );
39                              $result=$this->curhatmodel-
    >get_data("curhatan",$wherecurhat)->result();
40                              $data['curhat']=$result;
41                              $this->load->view('timeline',$data);
42                      }
                }
43              else{
44                      $this->load->view('login');
                }
        }
}
```

Penjelasan kode program:

28-44 Potongan kode digunakan untuk menjalankan fungsi *timeline*. Pada baris ke 29 sampai baris 44 yakni menjalankan fungsi yang sama seperti baris 13 sampai 26 yakni berfungsi menghindari untuk menampilkan halaman *login* ketika *user* sudah melakukan *login* dan barhasil masuk ke halaman timeline, jika *user* membuka website tersebut di tab baru.

5.2.4 *Class* Usermodel.php

Pada potongan kode program dibawah ini adalah kode dari kelas *model* yang dibutuhkan untuk membangun program *shared session* pada tabel 5.4.

Tabel 5.4 *Source-code Class* Usermodel.php

```
1   <?php
2   class Usermodel extends CI_Model{
3           function cek_login($table,$where){
4                   return $this->db->get_where($table,$where);
            }
```

Penjelasan kode program:

1-4 Potongan kode digunakan untuk *class* yang bernama *Pages* yang menge-*extend class* inti *Codeigniter* serta menjalankan fungsi *cek_login* dengan mencari variabel data pada tabel yang sesuai di *database* dan mengembalikan data yang sesuai jika ditemukan pada tabel tersebut.

```
5           function input_data($data){
6                   $this->db->insert('users',$data);
}
```

Penjelasan kode program:

5-6 Potongan kode digunakan untuk menjalankan fungsi *input_data* dengan memasukkan variabel data pada tabel '*users*' yang sesuai di *database*.

```
7   function get_data_single($table,$where){
8     $query= $this->db->get_where($table,$where);
9     if ($query->num_rows==1) {
10      return  $query->row();
      }
```

Penjelasan kode program:

7-10 Potongan kode digunakan untuk menjalankan fungsi *get_data_single* dengan mencari variabel data pada tabel yang sesuai di *database*. Jika baris pada tabel ditemukan maka akan mengembalikan nilai dari hasil tersebut.

```
11    else{
12      return false;
      } }}
```

Penjelasan kode program:

11-12 Potongan kode digunakan untuk jika variabel pada baris kosong atau tidak ditemukan maka akan mengembalikan nilai *boolean* salah.

5.3 Mekanisme *Shared Session* melalui tampilan halaman *website*

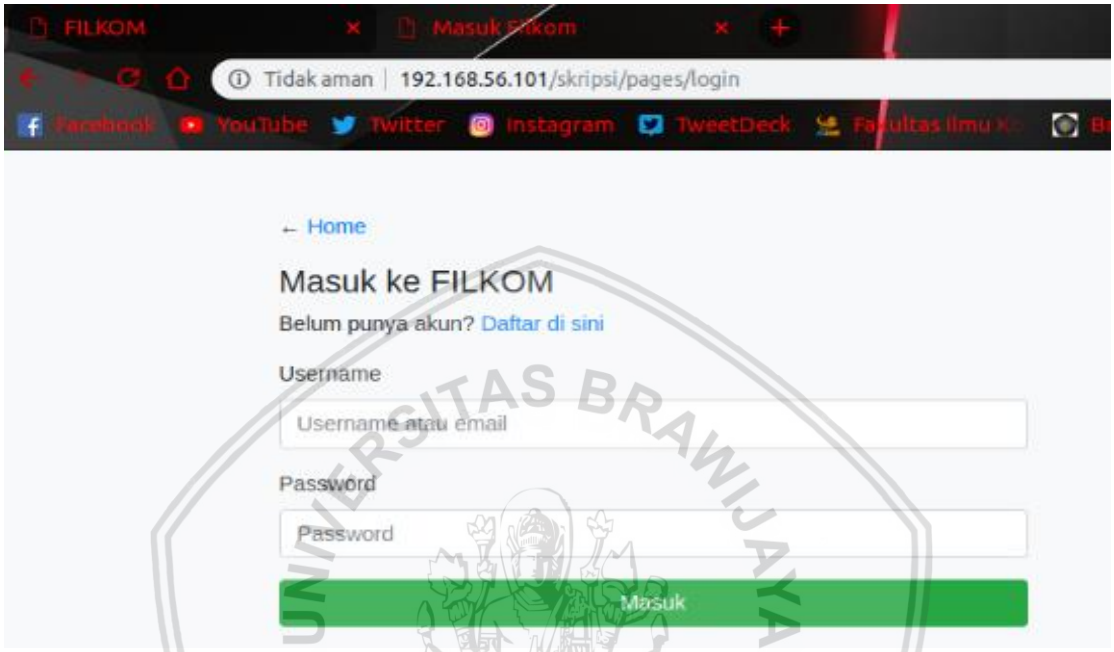
. Pada sub bab ini tampilan dari kode program setelah melakukan implementasi *shared session* berupa *website* serta digunakan untuk menguji sistem *shared session* pada penelitian ini dengan mempergunakan *PHP CodeIgniter* beserta *library* bawaan yang ada. Berikut ini beberapa tampilan aplikasi *website* yang sudah dirancang.

5.3.1 Halaman *Website* Sistem

Pada gambar ini menampilkan halaman utama *website* dengan memiliki fitur *login* dan *register* sebagai awal terbentuknya *session ID* yang didapatkan melalui *cookies* pada *browser*, seperti yang dapat dilihat pada gambar 5.17.

Gambar 5.17 Tampilan halaman utama dari aplikasi *website*

Pada gambar ini menampilkan halaman *login* sebagai awal mula autentikasi untuk *user* masuk ke dalam halaman *timeline* beserta terbentuknya *session* yang berisi *session ID* yang didapatkan melalui *cookies* pada browser, seperti yang dapat dilihat pada gambar 5.18.

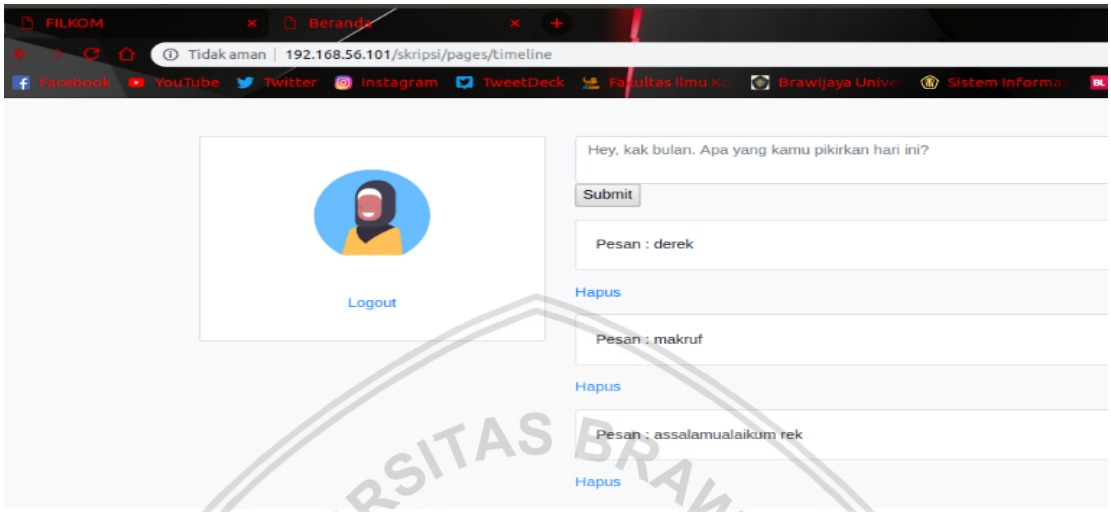


Gambar 5.18 Tampilan halaman *Login* dari aplikasi *website*

Pada gambar ini menampilkan halaman *register* yang berfungsi untuk *user* agar dapat mendaftarkan data diri sesuai ketentuan yang dibutuhkan *website* sehingga dapat melakukan *login* atau masuk ke dalam halaman *timeline*, tampilan dapat dilihat pada gambar 5.19.

Gambar 5.19 Tampilan halaman *register* dari aplikasi *website*

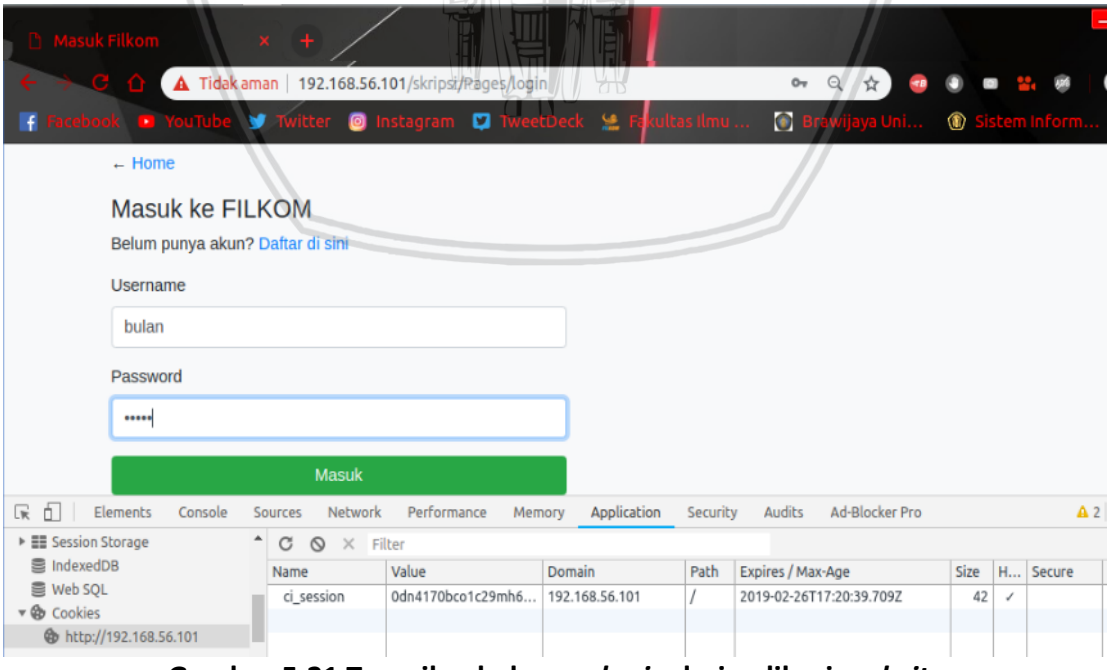
Pada gambar ini menampilkan halaman *timeline* untuk *user* yang berfungsi memberitahukan bahwa *user* tersebut telah berhasil masuk dan terverifikasi datanya sudah benar oleh *database*. Tampilan dapat dilihat pada gambar 5.20.



Gambar 5.20 Tampilan halaman *timeline* dari aplikasi *website*

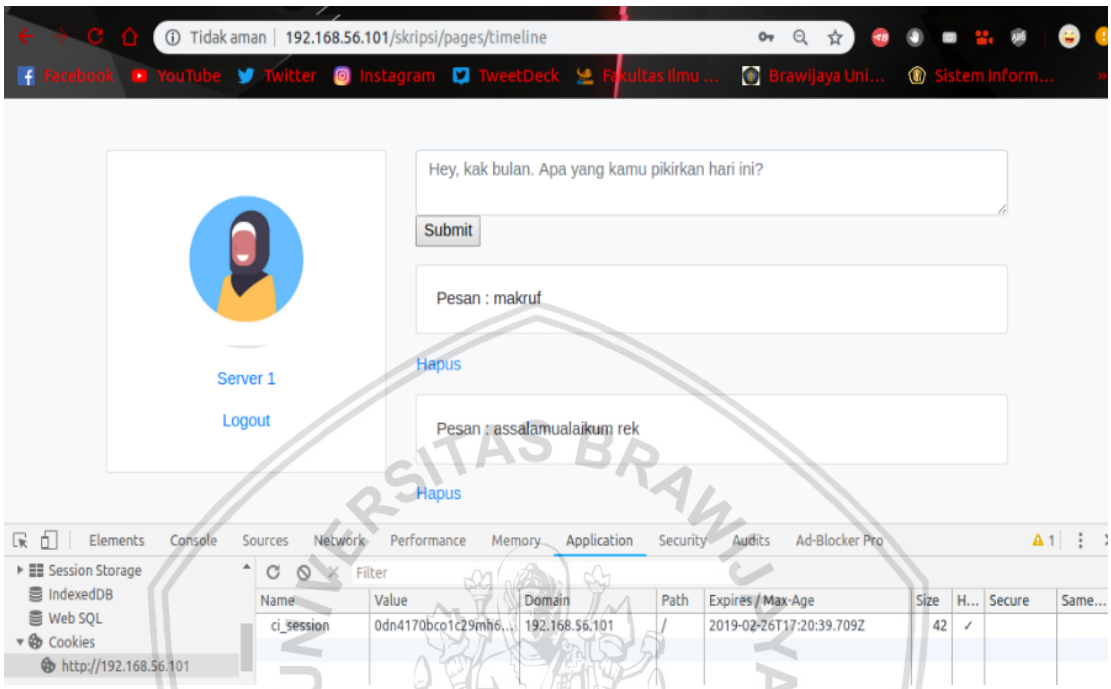
5.3.2 *Shared Session* pada halaman *timeline*

Pada gambar ini menampilkan halaman *login website*, dimana sebelumnya sudah terbangun sebuah *cookies* saat mengakses *website* pada *browser* dan menjadi cikal bakal *session* terbentuk sekaligus yang akan disimpan pada *database* seperti yang dapat dilihat pada gambar 5.21.



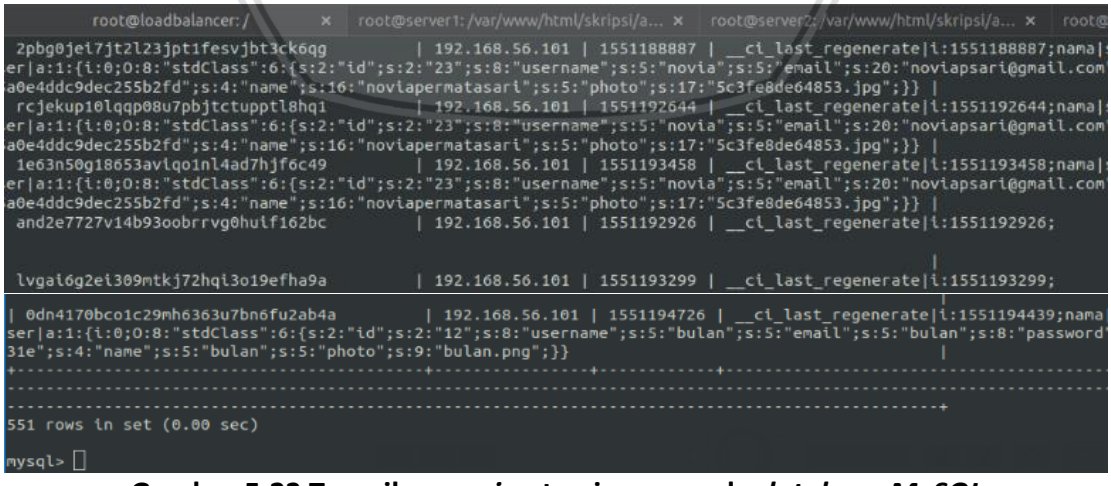
Gambar 5.21 Tampilan halaman *login* dari aplikasi *website*

Selanjutnya setelah melakukan *login* berhasil maka akan dialihkan halaman *timeline* serta menyimpan data *session* ke *database* berdasarkan *session ID* yang diberikan sewaktu meminta *cookies*, tampilan dapat dilihat pada gambar 5.22.



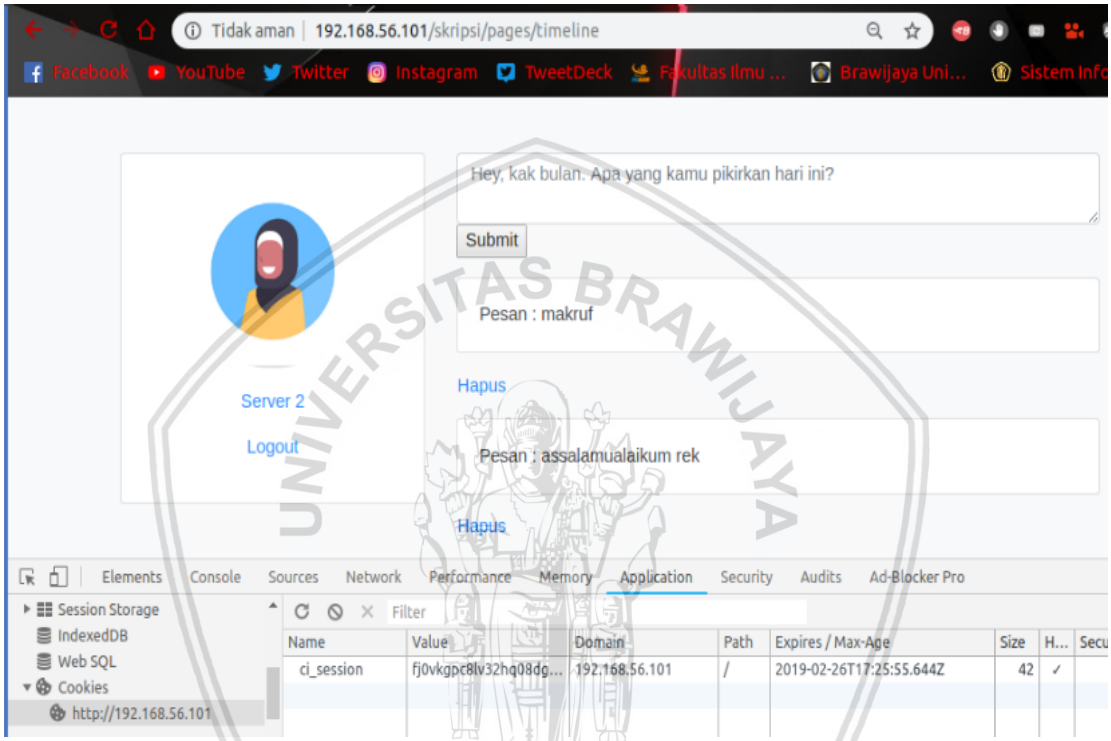
Gambar 5.22 Tampilan halaman *timeline* dan *cookies* pada *website*

Tampilan data *session* telah disimpan ke *database* yang diproses setelah *login* berhasil berdasarkan *session ID* yang didapatkan sesuai pada *cookies*, proses tersebut terjadi pada *database MySQL*. Dan seluruh *web server* yang aktif lainnya dapat mengakses *session* tersebut ke *database* secara berkala. Tampilan data dapat dilihat pada gambar 5.23.



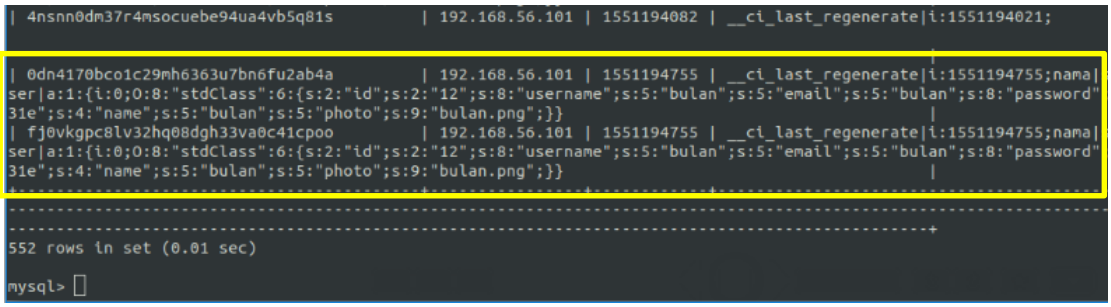
Gambar 5.23 Tampilan *session* tersimpan pada *database MySQL*

Kemudian setelah itu jika halaman *website* di *refresh* akan oleh *user* akan berpindah *server web* secara tidak langsung karena melakukan mekanisme *Round Robin* dilakukan oleh mesin dari aplikasi *load balancing* dan diwaktu bersamaan juga mengaktifkan mekanisme *shared session* karena *login* pertama kali tidak dilakukan pada *server web* ini, maka *server web* ini meminta session ke database agar dapat menampilkan data milik *user* yang aktif tersebut. Tampilan dapat dilihat pada gambar 5.24.



Gambar 5.24 Tampilan halaman *timeline* dan *cookies* pada *server web*

Tampilan data *session* baru telah berhasil dibagikan untuk *server web* yang membutuhkan data *session* diwaktu yang sama untuk *user*, proses tersebut terjadi pada *database MySQL*. Tampilan data dapat dilihat pada gambar 5.25.



Gambar 5.25 Tampilan data *session* pada *database MySQL*

Jadi dengan adanya mekanisme tersebut dapat memudahkan *user* tidak perlu *login* kembali meskipun mengakses *server web* yang berbeda di waktu yang sama, proses tersebut terus berulang karena terdapat mekanisme Round Robin yang aktif pada *load balancing*. Penjabaran mengenai mekanisme diatas merupakan gambaran dari salah satu *server load balancing* yakni pada *server* 192.168.56.102 ke *server* 192.168.56.103 melalui *session* yang diminta pada *server database MySQL* dan untuk *server* aktif yang lain tidak jauh berbeda mekanisme permintaan *session* nya.

Mekanisme *shared session* ini akan berakhir secara otomatis jika *user* tidak aktif sama sekali pada halaman *website* selama 1800 detik atau 30 menit dan *cookies* akan dihapus oleh sistem kemudian digantikan *cookies* baru, di waktu yang bersamaan juga *database* akan menghapus data *session* saat itu. Bilamana *user* melakukan *logout* akan dialihkan ke halaman utama *website* sekaligus akan menampilkan *cookies* baru dan menghapus seluruh data *session* yang sebelumnya telah tersimpan.

BAB 6 PENGUJIAN SISTEM

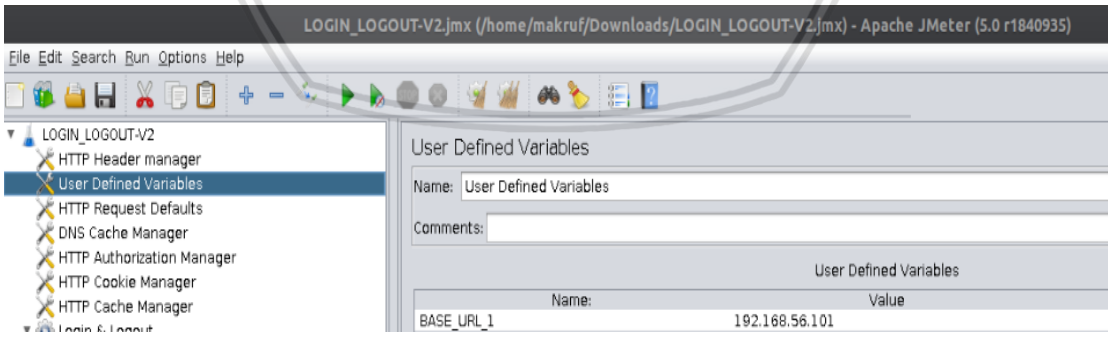
Pada bab ini menjelaskan tentang pengujian dari implementasi sistem yang sudah diterapkan. Hasil dari pengujian ini diperlukan untuk mengetahui apakah dari sistem sudah sesuai dengan kebutuhan dan mengetahui kinerja dari aplikasi. Pengujian akan dilakukan berdasarkan perancangan pengujian yang telah dijabarkan pada bab sebelumnya.

6.1 Pengujian *Login* dan *Logout* menggunakan *Shared Session*

Pengujian kinerja sistem dilakukan aplikasi uji *Jmeter* yang bertujuan untuk mengetahui apakah aplikasi sudah sesuai dengan kebutuhan yang telah dirancang. Skema pengujian dijalankan melalui fitur pada aplikasi *website* yang sudah diimplementasikan sebelumnya. Skema pengujian ini dilakukan untuk mengetahui kinerja *shared session* yang berada dalam klaster *server web* yang dimulai dari mengakses *website* dan *user* masuk ke dalam aplikasi *website* atau melakukan *login* hingga melakukan *logout*.

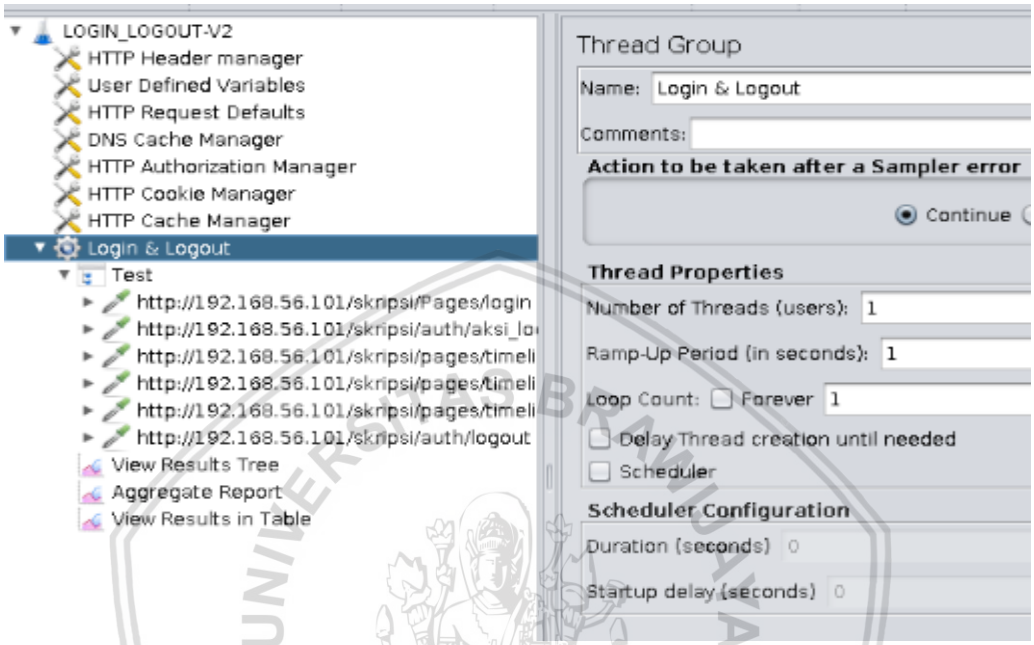
6.1.1 Pengujian *Login* menggunakan *Shared Session*

Dalam pengujian ini dibuat *user virtual* sebanyak 1 *user virtual* yang sudah didaftarkan sebelumnya datanya ke *database* dan dikondisikan melakukan *login* kedalam *website* dan aplikasi *load balancing* akan mengarahkan ke salah satu *server web* agar menampilkan *website* yang diminta. Namun setelah *login* berhasil *server web* yang sudah diakses dari arahan *load balancing* tersebut mengalami *server down* atau gagal menyediakan layanan untuk *user*. Pada sub bab ini akan dijelaskan langkah-langkah dari pengujian *login* melalui tampilan aplikasi Jmeter. Tampilan konfigurasi dapat dilihat pada gambar 6.1.



Gambar 6.1 Tampilan konfigurasi alamat *URL* pada aplikasi uji

Dalam rangkaian pengujian ini dilakukan oleh 1 *user virtual* yang mengakses *website* dalam waktu 1 detik dikondisikan melakukan *login* kedalam *website* sampai dengan sampai melakukan proses *logout* dan jika berhasil dapat masuk ke dalam halaman *timeline*. Tampilan konfigurasi dapat dilihat pada gambar 6.2.

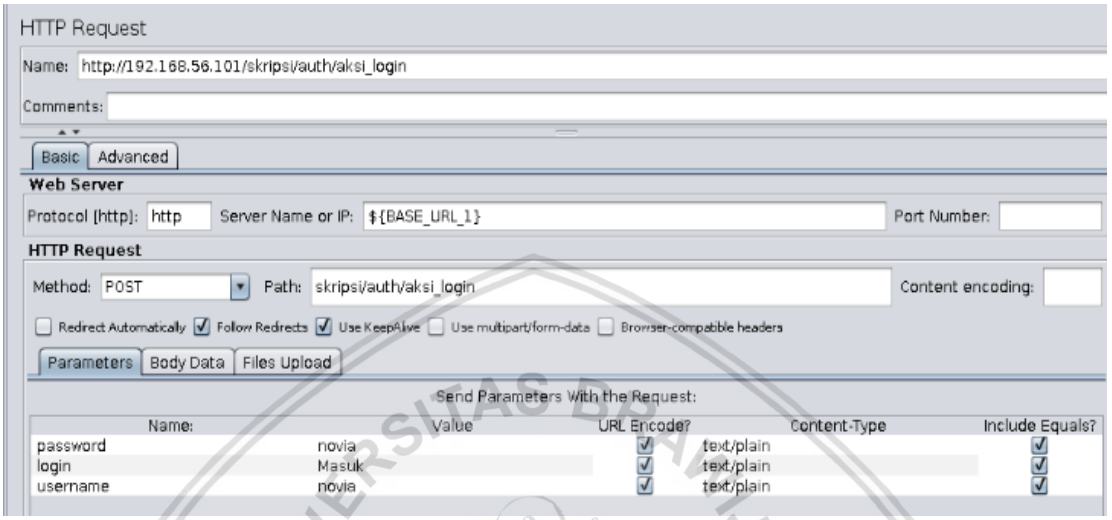


Gambar 6.2 Tampilan alur dari rangkaian *login* pada aplikasi uji

Kemudian menambahkan konfigurasi *HTTP request* pada *thread* yang berfungsi untuk mengakses *website* dalam membuka halaman *login* kedalam *website* melalui *IP server load balancing*. Tampilan konfigurasi dapat dilihat pada gambar 6.3.

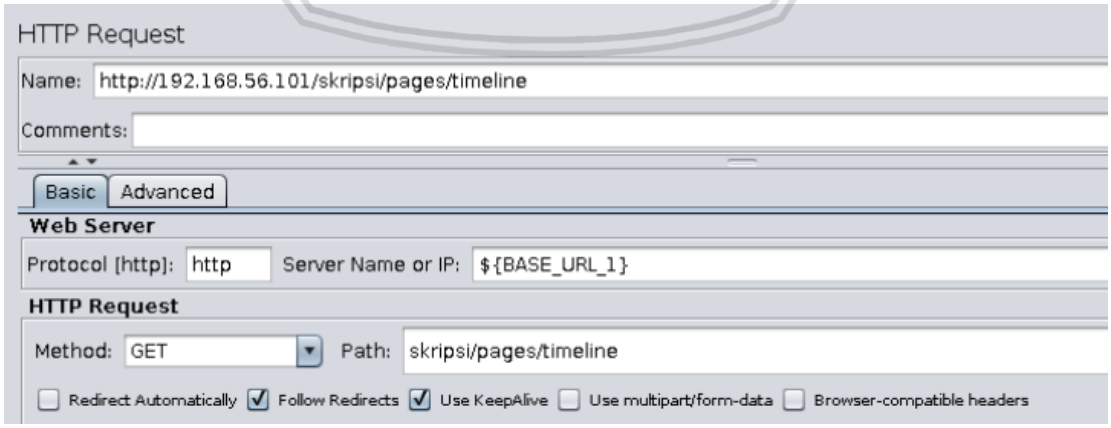
Gambar 6.3 Tampilan *HTTP request* halaman login pada aplikasi uji

Menambahkan konfigurasi *HTTP request* pada *thread* yang berfungsi untuk mengakses *website* dalam memproses data masukan dari *user* berupa *username* dan *password* pada halaman *login* kedalam *website* melalui *IP server load balancing*. Tampilan konfigurasi dapat dilihat pada gambar 6.4.

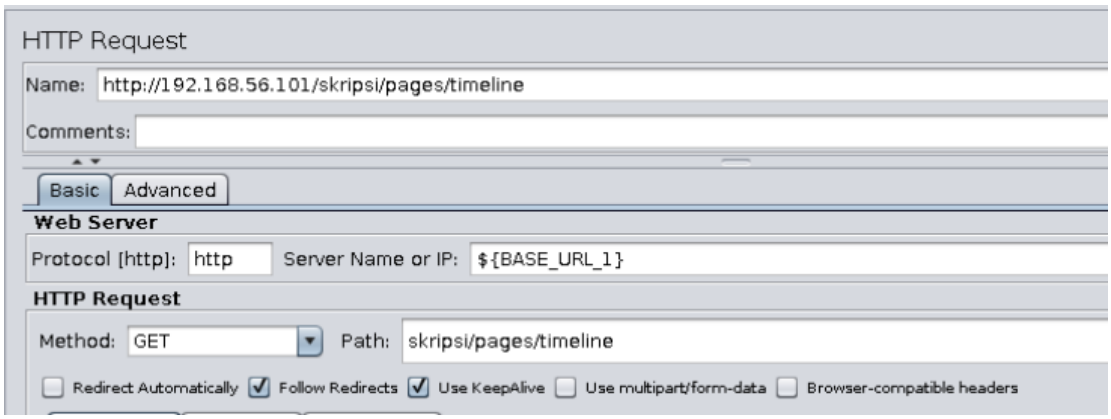


Gambar 6.4 Tampilan proses data dari *login* pada aplikasi uji

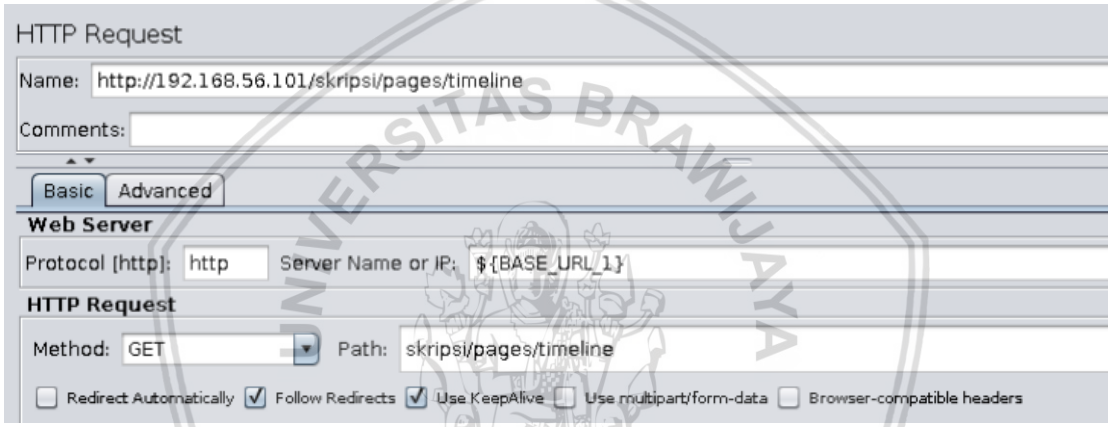
Menambahkan konfigurasi *HTTP request* pada *thread* yang berfungsi untuk menampilkan data *user* berupa tampilan halaman *timeline* yang telah melakukan autentikasi *login* kedalam *website* melalui IP *server load balancing* dan terbentuk sebuah *session*. Terdapat 3 konfigurasi *HTTP request* karena peneliti mengaktifkan 3 *server web* dalam satu klaster *server* sebagai sarana penyedia layanan *request* dan *response* dari *Apache*. Setiap *user* setelah melakukan *login* akan secara acak masuk ke salah satu *server web* yang diarahkan *load balancing*. Tampilan konfigurasi dapat dilihat pada gambar 6.5 sampai dengan gambar 6.7.



Gambar 6.5 Tampilan konfigurasi *HTTP request* halaman *timeline*

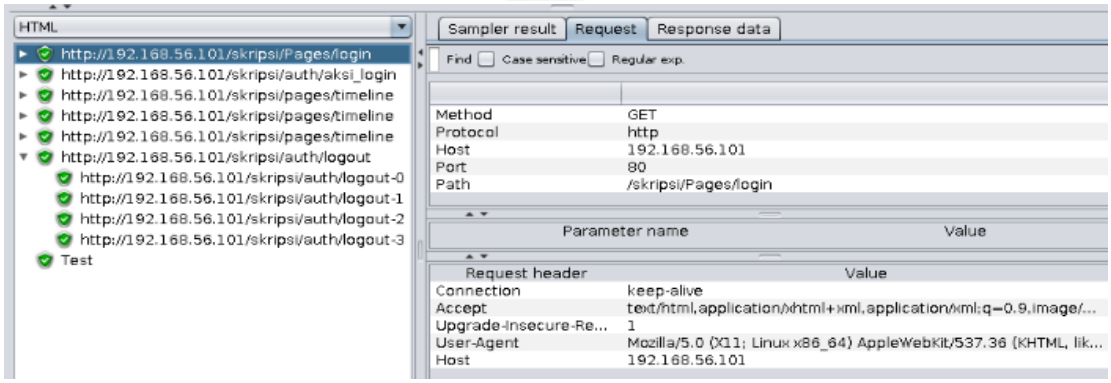


Gambar 6.6 Tampilan konfigurasi *HTTP request* halaman *timeline*



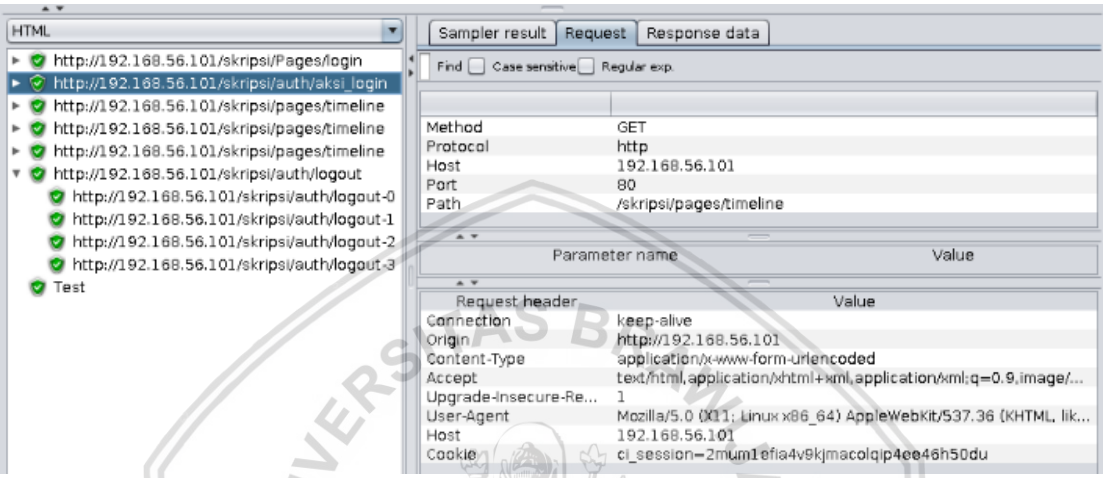
Gambar 6.7 Tampilan konfigurasi *HTTP request* halaman *timeline*

Saat pengujian ini dilakukan, di waktu yang sama sistem otomatis masuk ke dalam *server* secara acak sebagai autentikasi *login* akses pertama. Lalu peniliti menguji dengan mematikan aplikasi *web server* saat akses *login* pertama dilakukan oleh *user*. Maka terlihat dampak perpindahan dari *server web* yang mati tersebut dikelola oleh *server web* aktif lainnya. Dalam pengujian langkah awal yakni proses membuka halaman *login*. Tampilan konfigurasi dapat dilihat pada gambar 6.8.

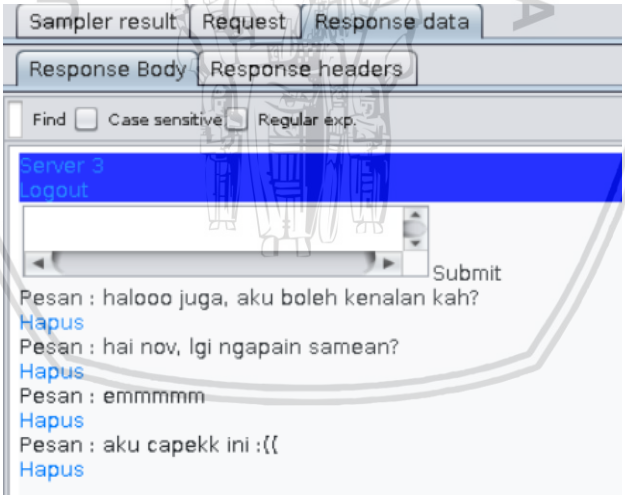


Gambar 6.8 Tampilan *request header* halaman *login*

Setelah halaman *login* dapat dibuka data *user virtual* dimasukkan oleh aplikasi uji dan mendapatkan *cookies* yang akan disimpan sebagai data *session* dari *user* yang aktif saat itu autentikasi *login* akses pertama. Karena menggunakan algoritma *Round Robin* yang secara acak mendistribusikan trafik dan saat proses melakukan *login* bertepatan pada *server web* yang ketiga. Tampilan dapat dilihat pada gambar 6.9 dan gambar 6.10.

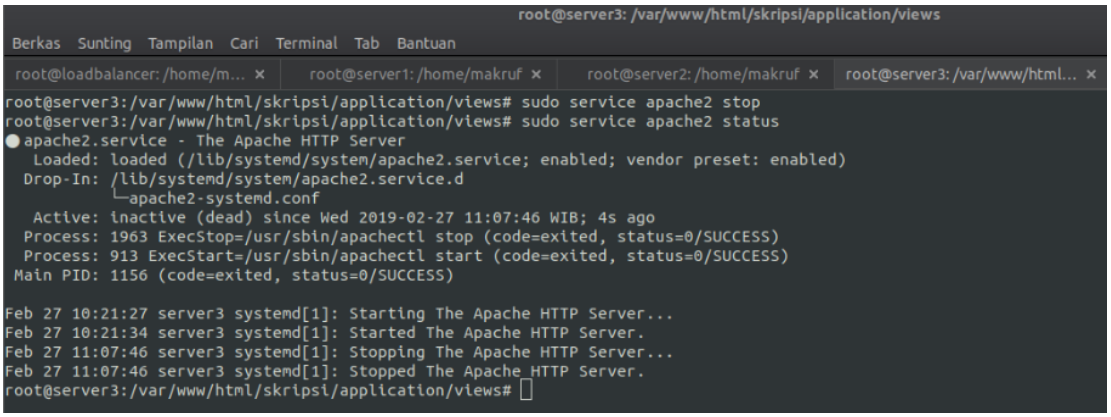


Gambar 6.9 Tampilan *request header* proses data saat *login*



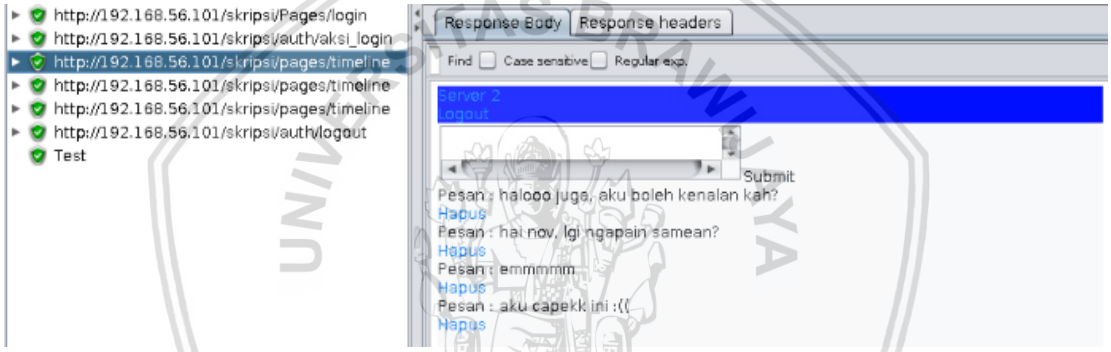
Gambar 6.10 Tampilan *response* data saat *login*

Selanjutnya menonaktifkan *service Apache* dari *server web* yang ketiga. Karena autentikasi *login* akses pertama dilakukan pada *server web* yang ketiga. Tampilan *service* yang sudah nonaktif dapat dilihat pada gambar 6.11.



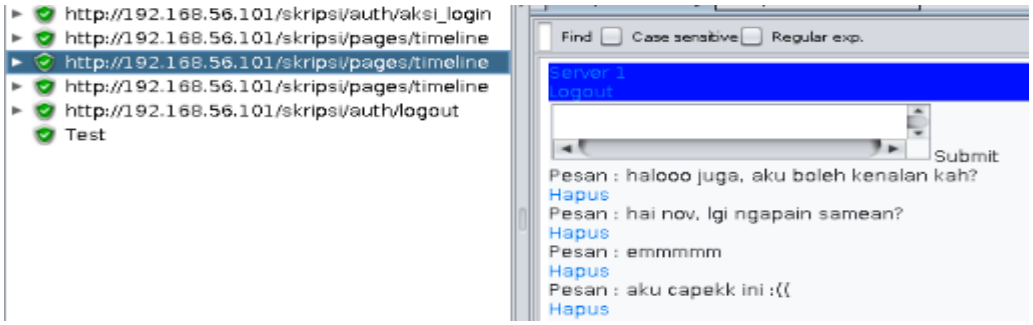
Gambar 6.11 Tampilan *service Apache* di *server web* ketiga

Kemudian menampilkan halaman *timeline* dari masing-masing aplikasi *web server*. Tampilan *response* data dapat dilihat pada gambar 6.12.

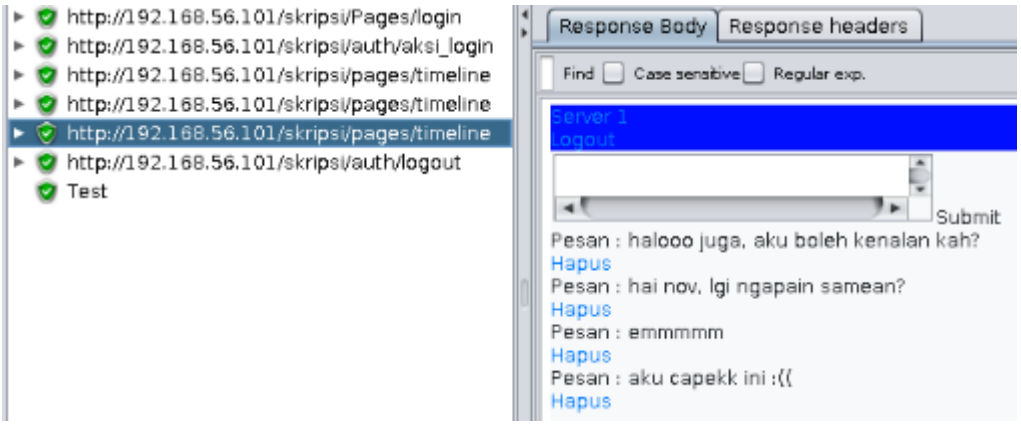


Gambar 6.12 Tampilan halaman *timeline* dari *server web* kedua

Pembagian trafik secara berurutan dilakukan oleh *load balancing* yakni seharusnya dilanjutkan pada *server web* ketiga, namun *server web* yang ketiga aplikasi *Apache* telah dihentikan *service* nya maka tidak dapat menampilkan halaman *timeline*. Maka secara otomatis pada *request* akses halaman *timeline* yang lain dikelola oleh *server web* pertama. Tampilan *response* data dapat dilihat pada gambar 6.13 dan gambar 6.14.



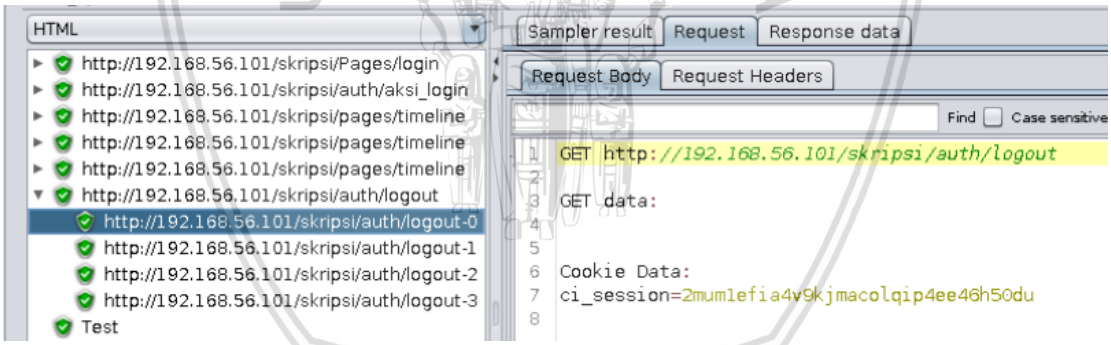
Gambar 6.13 Tampilan halaman *timeline* dari *server web* pertama



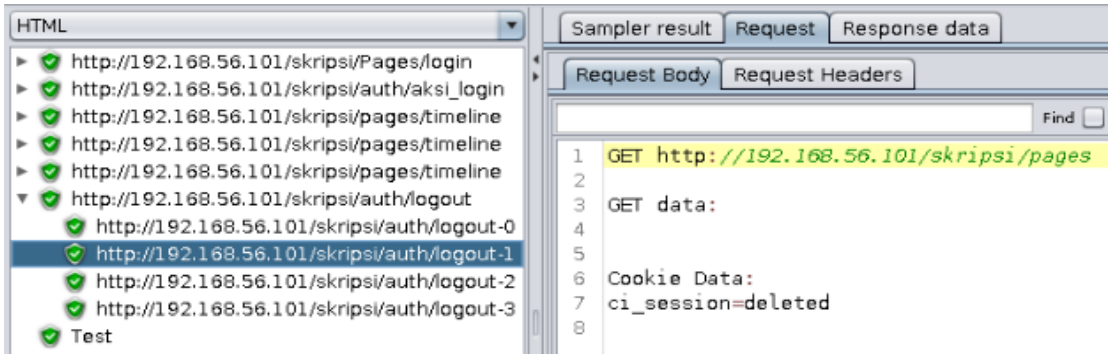
Gambar 6.14 Tampilan halaman *timeline* selanjutnya di *server web* pertama

6.1.2 Pengujian *Logout* bedasarkan penggunaan *Shared Session*

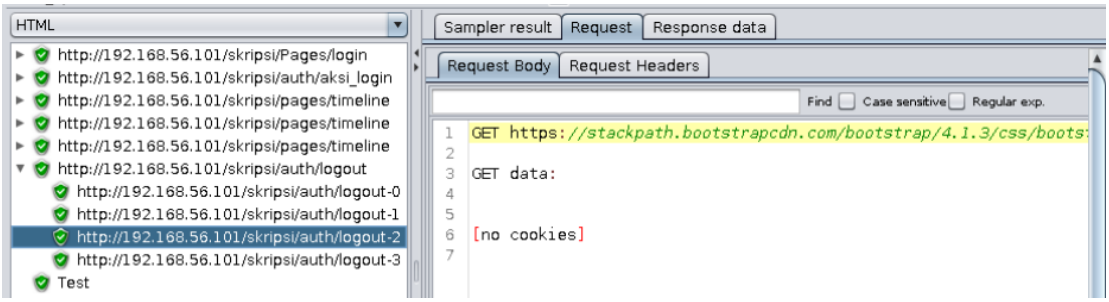
Dalam pengujian ini dilakukan bertujuan untuk mengetahui bahwa *session* yang sudah diakses oleh seluruh *web server* atau dibagikan dari satu *web server* ke *web server* yang lain dalam satu klaster telah dipastikan sudah dihapus dan diakhiri dengan diganti *session ID* baru saat terjadinya proses *request* perintah untuk *logout* dari *user*. Tampilan proses *logout* dapat dilihat pada gambar 6.15 sampai dengan gambar 6.18



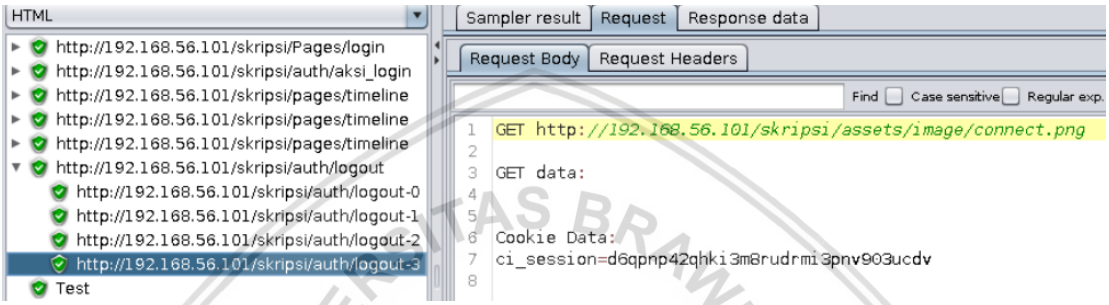
Gambar 6.15 Tampilan proses *logout* bedasarkan *session ID*



Gambar 6.16 Tampilan konfirmasi *cookies* telah terhapus



Gambar 6.17 Tampilan konfirmasi *cookies* kosong



Gambar 6.18 Tampilan *cookies* baru setelah *logout*

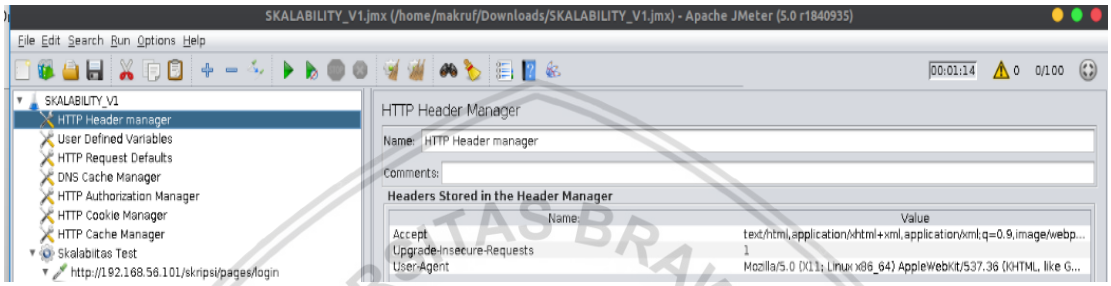
Bedasarkan rangkaian pengujian login sampai dengan logout yang sudah dilakukan dapat disimpulkan bahwa sistem *shared session* telah berhasil diujikan melalui skema pengujian ini dengan parameter yang dapat dilihat pada tabel 6.1.

Tabel 6.1 Hasil pengujian kinerja aplikasi *website*

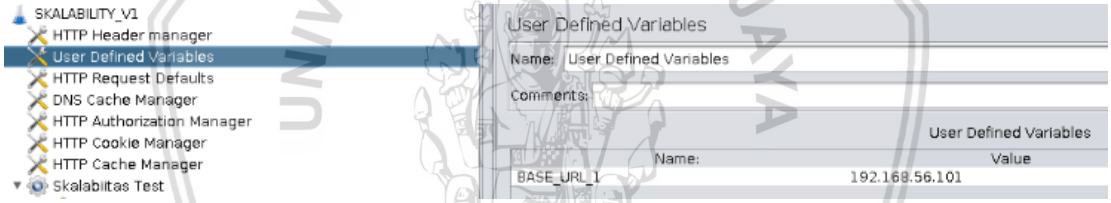
No.	Pengujian Kinerja	Parameter Keberhasilan	Status
1	Aplikasi website menjalankan fitur *login* di ketiga *web server*.	Aplikasi website berhasil diakses oleh *user* dan dapat membuka halaman *timeline* di ketiga *web server*.	**VALID**
2	Aplikasi website menjalankan sistem *shared session*.	Aplikasi website berhasil menjalankan sistem *shared session* dengan baik dengan melakukan perpindahan *web server* saat salah satu *web server* dalam keadaan mati.	**VALID**
3	Aplikasi website menjalankan fitur *logout*.	Aplikasi website berhasil menghapus *sesson* yang aktif saat user telah melakukan *logout*.	**VALID**

6.2 Pengujian Black box dari sistem *Shared Session*

Pengujian black box dilakukan melalui aplikasi uji *Jmeter* yang bertujuan untuk mengetahui skalabilitas sistem dari aplikasi *website* apakah sudah sesuai dengan kebutuhan yang telah dirancang. Skema pengujian dijalankan melalui tiga skenario yang sudah direncanakan pada bab perancangan pengujian sebelumnya. Sebelum melakukan pengujian skalabilitas ada berberapa hal yang perlu dikonfigurasi pada aplikasi uji agar pengujian sesuai dengan masing-masing skenario, yakni dapat dilihat pada gambar 6.19 dan 6.20.



Gambar 6.19 Tampilan konfigurasi *HTTP header*



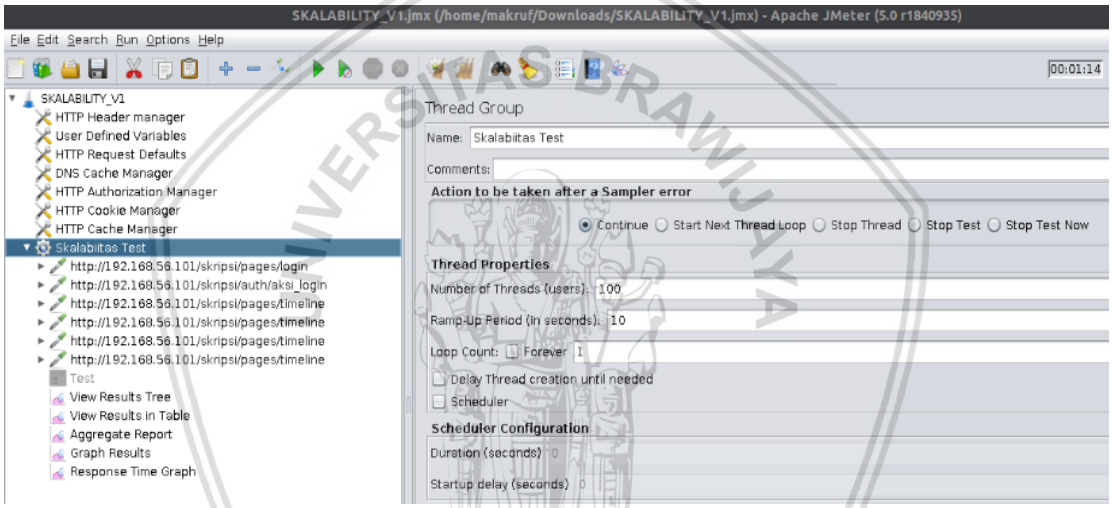
Gambar 6.20 Tampilan konfigurasi alamat *URL* yang akan diakses

Adapun konfigurasi yang penting lainnya untuk menjalankan fungsi skenario dalam aplikasi *Jmeter* ini, yakni:

1. *HTTP Request Defaults* berfungsi untuk menetapkan nilai *default* dari setiap *request* yang dilakukan.
2. *DNS Cache Manager* berfungsi sebagai yang menerima konten dari IP yang berbeda karena ada beberapa *server web* aktif di belakang *load balancer*.
3. *HTTP Authorization Manager* berfungsi untuk menentukan satu atau lebih *login* pengguna pada halaman *web* yang dibatasi menggunakan otentikasi *server*.
4. *HTTP Cookie Manager* berfungsi untuk mengelola dalam menyimpan *cookie* dan digunakan untuk semua permintaan di masa mendatang ke situs *web* tertentu.
5. *HTTP Cache Manager* berfungsi untuk menambahkan fungsionalitas *caching* ke permintaan *HTTP* dalam ruang lingkupnya untuk mensimulasikan fitur *cache* browser.

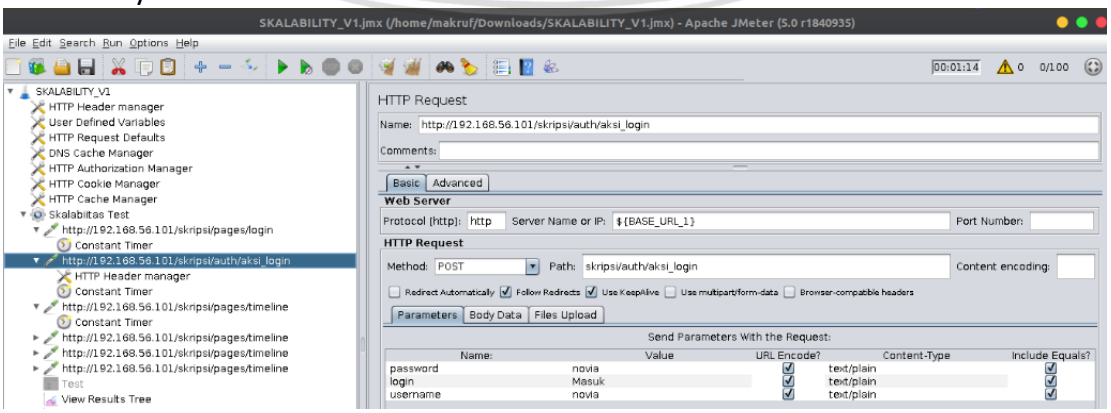
6.2.2 Skenario ke-1 sistem *Shared Session*

Pada skenario yang pertama ini melakukan *request session* untuk dilihat dari seberapa handal dalam mendistribusikan *session* yang dibutuhkan. Skenario ini menggunakan 100 *user virtual* yang akan menjalankan rangkaian skenario dengan melakukan *request session* ke *database* dan diberikan *session ID* yang berbeda-beda agar mengetahui kinerja dari sistem *shared session*. Dalam skenario ini dikondisikan ada 1 *server web* yang dinonaktifkan dari 4 *server web* yang aktif. Pengujian dimulai dengan 100 *user* data melakukan *login* ke aplikasi *website* melalui *load balancing server* dengan IP 192.168.56.101 pada 2 Maret 2019 pukul 19:00. Dari 100 *user* akan secara acak *login* ke dalam aplikasi *website* yang akan dikelola *load balancing* yang berisi 4 *server web* aktif dan disetiap 1 *user* melakukan *request* akan diberikan jeda waktu antar *user* lainnya dengan waktu total sebesar 10 detik.



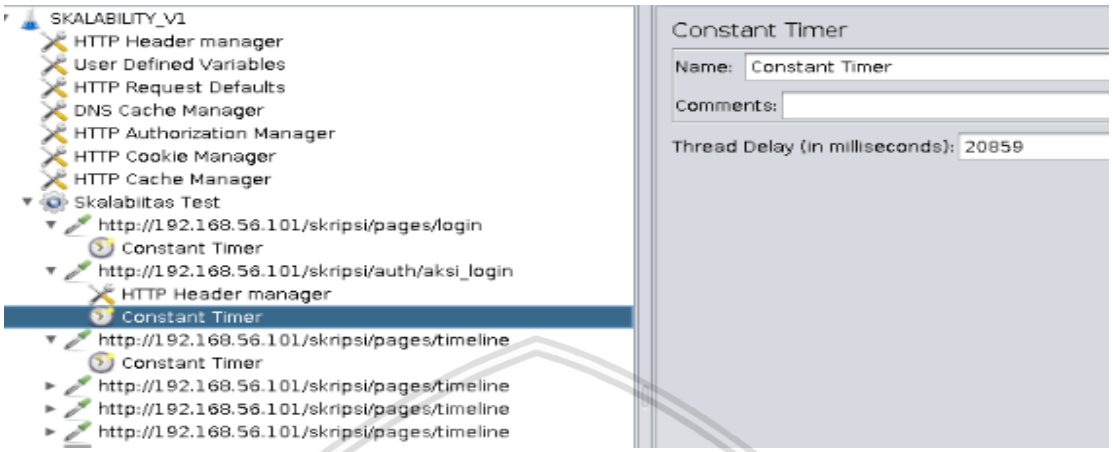
Gambar 6.21 Tampilan konfigurasi *thread* pada skenario ke-1

Dalam konfigurasi *request login* telah ditentukan satu *user* yang sama akan melakukan 100 kali setiap *request HTTP* bedasarkan konfigurasi pada *thread* sebelumnya.



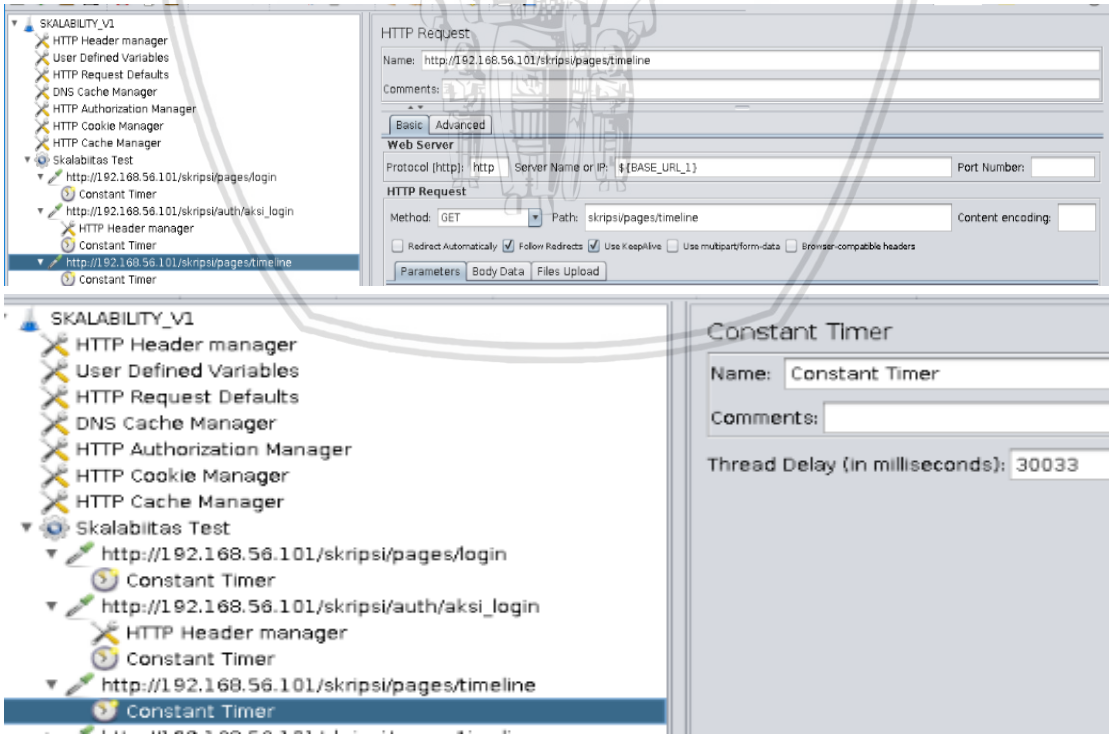
Gambar 6.22 Tampilan konfigurasi *request* autentikasi *login*

Penentuan waktu *delay* yang bertujuan untuk agar seluruh *request* bisa dieksekusi secara keseluruhan pada proses autentikasi *login*.



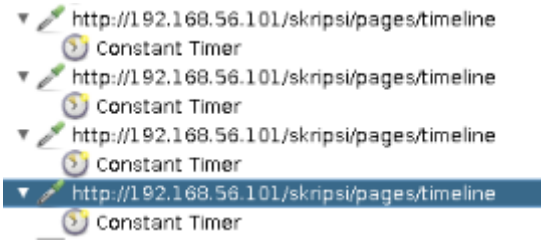
Gambar 6.23 Tampilan waktu *delay request* autentikasi *login*

Konfigurasi *request* untuk menampilkan halaman *timeline* dan penentuan waktu *delay*. Waktu *delay* diberikan sebesar 30033 milidetik atau sebesar 30,033 detik. Waktu diberikan bertujuan untuk sebelum *request* dieksekusi salah satu dari keempat *web server* akan dimatikan.



Gambar 6.24 Tampilan *request* halaman *timeline* dan waktu *delay*

Daftar konfigurasi *request* untuk menampilkan halaman *timeline*. Dikarenakan yang di*request* dari empat *server web* yang berbeda, maka harus menambahkan konfigurasi untuk *HTTP request* halaman *timeline* disetiap *server web* yang diakses.

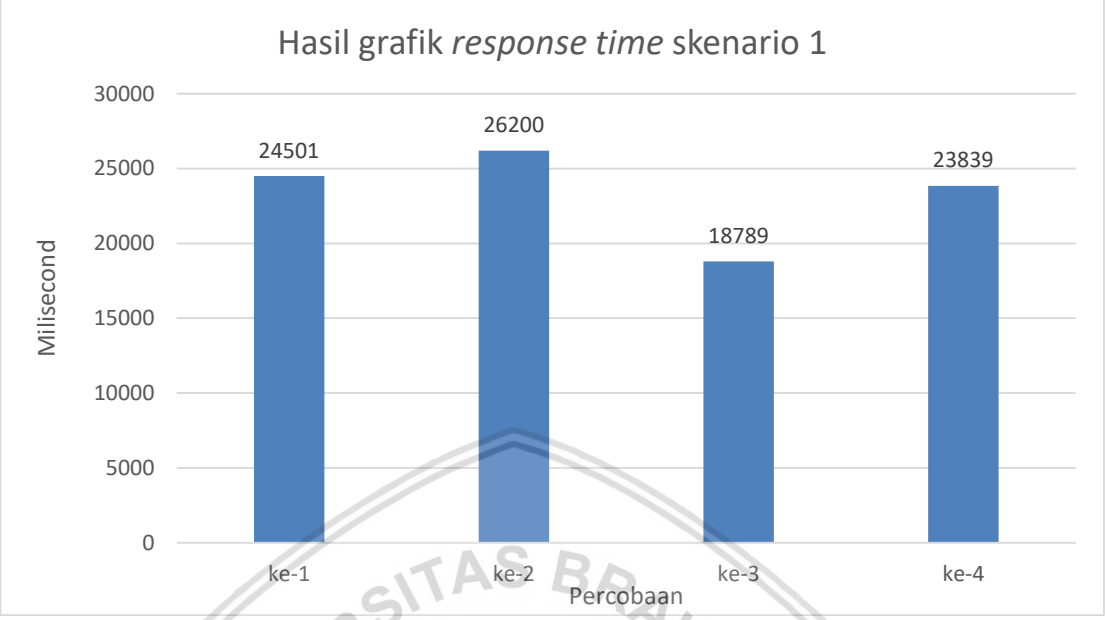


Gambar 6.25 Tampilan request halaman timeline seluruh server web

Dalam skenario ini dilakukan empat kali percobaan untuk menguji kehandalan setiap masing-masing *server web* nya. Yang pertama dengan menonaktifkan layanan dari aplikasi Apache *server web* satu, kemudian jika sudah selesai pecobaan pertama dilanjutkan menonaktifkan aplikasi *Apache server web* kedua dan ini dilakukan hingga percobaan keempat. Saat skenario dalam proses eksekusi dan proses tersebut terjadi setelah 100 data *user* selesai melakukan proses *login* lalu sebelum masuk ke halaman timeline, peneliti menonaktifkan layanan aplikasi *Apache* dari *server web* secara manual. Berikut hasil data dari empat kali percobaan melalui aplikasi pengujian.

Tabel 6.2 Tabel hasil *response time* dari skenario ke-1

Percobaan	*Request*	Hasil *server web* dalam *response time* (ms)				Total Hasil Percobaan	Status *server web* bedasarkan percobaan	Keberhasilan
		1	2	3	4			
Ke-1	400	0	5267	11134	8100	24501	*Server web* 1 mati	Tidak ada *erorr*
Ke-2	400	5200	0	11000	10000	26200	*Server web* 2 mati	Tidak ada *erorr*
Ke-3	400	9850	4634	0	8167	18789	*Server web* 3 mati	Tidak ada *erorr*
Ke-4	400	8100	10000	4300	0	23839	*Server web* 4 mati	Tidak ada *erorr*



Gambar 6.26 Hasil grafik *response time* dari skenario ke-1

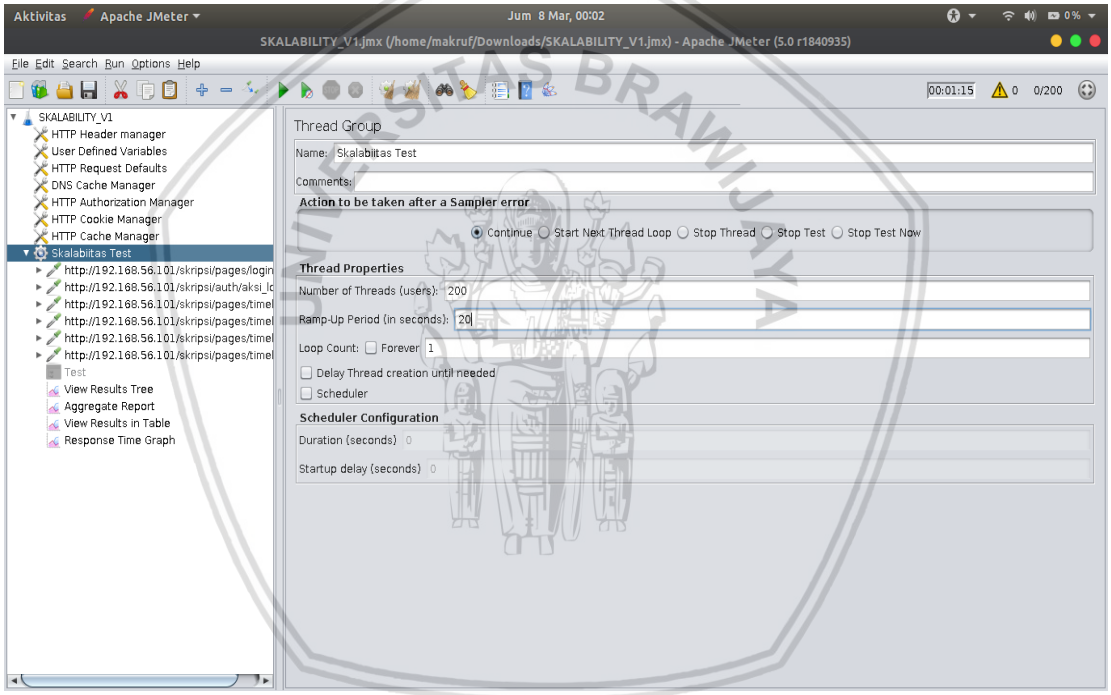
Berdasarkan hasil skenario yang ditampilkan melalui hasil gambar grafik dan hasil tabel dengan waktu tempuh dalam empat kali percobaan dari masing-masing *server web* didapatkan waktu rata-rata selama 01 menit 20 detik dalam menyelesaikan setiap percobaan. Terjadi perubahan waktu respon dari 100 data *user* acak yang telah *login* berhasil namun peningkatan waktu respon dalam rentang waktu ± pada detik ke 64 hingga detik ke 80 dari total rata-rata waktu tanggap yang dicapai, perubahan tersebut disebabkan salah satu *server web* yang nonaktif dan permintaan *session* di *handle* oleh *server web* yang lain secara acak. Dengan hasil ini maka menunjukkan bahwa aplikasi uji berhasil menjalankan skenario yang pertama. Pada skenario ini saat menjalankan *share session* dengan aktif atau non aktifnya dari salah satu *server web* akan mempengaruhi waktu respon pada kinerja sebuah *website*. Untuk detail seluruh hasil tabel dan grafik yang berhasil dari aplikasi pada pengujian skenario satu dapat dilihat pada Lampiran A.1

Tabel 6.3 Tabel rata-rata *response time* dari skenario ke-1

Percobaan	*Server Web* Aktif	*Request*	Rata rata *response time* (ms)	Rata rata *response time* (sec)	Status *server web*
Ke-1	3	400	24501 milisecond	24,5 second	*Server web* 1 mati
Ke-2	3	400	26200 milisecond	26,2 second	*Server web* 2 mati
Ke-3	3	400	18789 milisecond	18,7 second	*Server web* 3 mati
Ke-4	3	400	23839 milisecond	23,8 second	*Server web* 4 mati

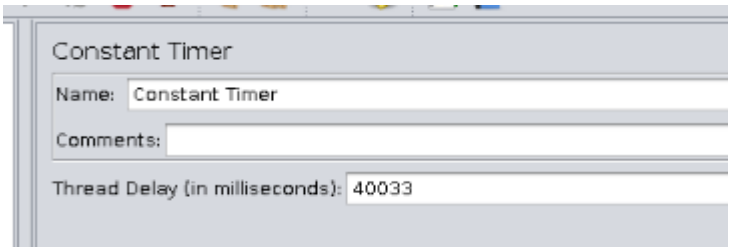
6.2.3 Skenario ke-2 sistem *Shared Session*

Pada skenario yang kedua ini melakukan *request session* untuk dilihat dari seberapa handal dalam mendistribusikan *session* yang dibutuhkan. Skenario ini menggunakan 200 *user virtual* yang akan menjalankan rangkaian skenario dengan melakukan *request session* ke *database* dan diberikan *session ID* yang berbeda-beda agar mengetahui kinerja dari sistem *shared session*. Dalam skenario ini dikondisikan ada 2 *server web* yang dinonaktifkan dari 4 *server web* yang aktif. Pengujian dimulai dengan 200 *user* data melakukan *login* ke aplikasi *website* melalui *load balancing server* dengan IP 192.168.56.101 pada 8 Maret 2019 pukul 00:02. Dari 200 *user* akan secara acak *login* ke dalam aplikasi *website* yang akan dikelola *load balancing* yang berisi 4 *server web* aktif dan disetiap 1 *user* melakukan *request* akan diberikan jeda waktu antar *user* lainnya dengan total waktu sebesar 20 detik.



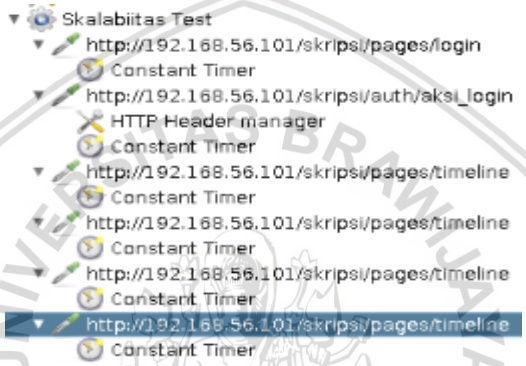
Gambar 6.27 Tampilan konfigurasi *thread* pada skenario ke-2

Konfigurasi *request* untuk penentuan waktu *delay* saat mengakses dan menampilkan halaman *timeline*. Waktu *delay* diberikan sebesar 40033 milidetik atau sebesar 40,033 detik. Waktu *delay* diberikan bertujuan untuk memberi jeda waktu sebelum *request* dieksekusi kemudian salah satu dari keempat *web server* akan dimatikan.



Gambar 6.28 Tampilan waktu *delay* untuk *request* halaman timeline

Untuk pengaturan dan daftar konfigurasi *request* pada skenario kedua tidak jauh berbeda dengan skenario yang pertama. Perbedaan yang signifikan hanya pada jumlah *user* data dan delay waktu tiap *user*.



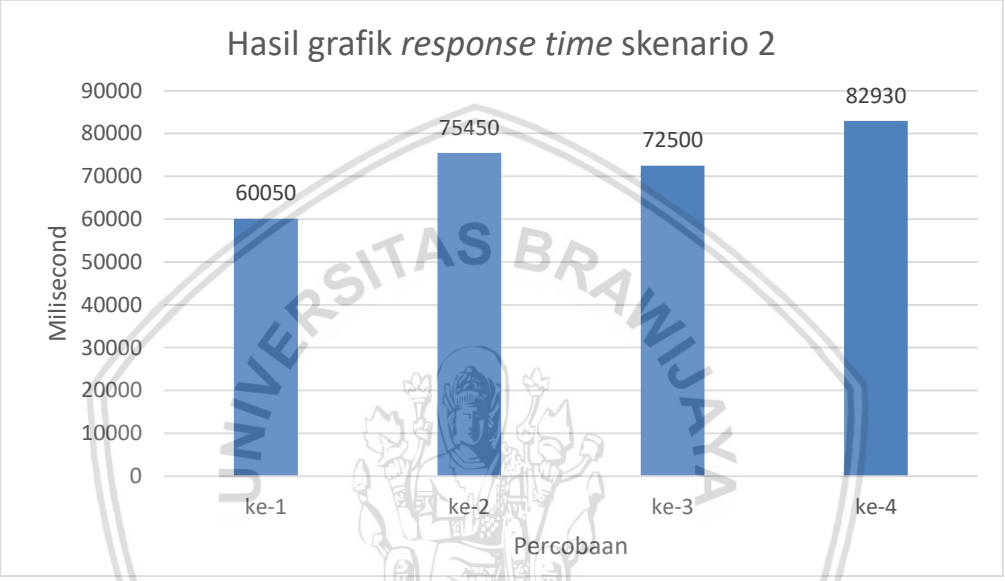
Gambar 6.29 Tampilan HTTP request pada skenario ke-2

Dalam skenario ini dilakukan empat kali percobaan untuk menguji kehandalan setiap masing-masing *server web* nya. Yang pertama dengan menonaktifkan layanan dari aplikasi *Apache server web* satu dan dua, kemudian jika sudah selesai pecobaan pertama dilanjutkan menonaktifkan aplikasi *Apache server web kedua* dan ketiga lalu ini dilakukan hingga percobaan keempat. Saat skenario dalam proses eksekusi dan proses tersebut terjadi setelah 200 data *user* selesai melakukan proses *login* lalu sebelum masuk ke halaman *timeline*, peneliti menonaktifkan layanan aplikasi *Apache* dari *server web* secara manual. Berikut hasil data dari empat kali percobaan melalui aplikasi pengujian.

Tabel 6.4 Tabel hasil *response time* dari skenario ke-2

Percobaan	*Request*	Hasil *server web* dalam *response time* (ms)				Total Hasil Percobaan	Status *server web* bedasarkan percobaan	Keberhasilan
		1	2	3	4			
Ke-1	800	0	0	36150	23900	60050	*Server web* 1 dan 2 mati	Tidak ada *erorr*
Ke-2	800	39450	0	0	36000	75450	*Server web* 2 dan 3 mati	Tidak ada *erorr*

Percobaan	*Request*	Hasil *server web* dalam *response time* (ms)				Total Hasil Percobaan	Status *server web* bedasarkan percobaan	Keberhasilan
		1	2	3	4			
Ke-3	800	42000	30500	0	0	72500	*Server web* 3 dan 4 mati	Tidak ada *erorr*
Ke-4	800	0	49370	33560	0	82930	*Server web* 1 dan 4 mati	Tidak ada *erorr*



Gambar 6.30 Hasil grafik *response time* dari skenario ke-2

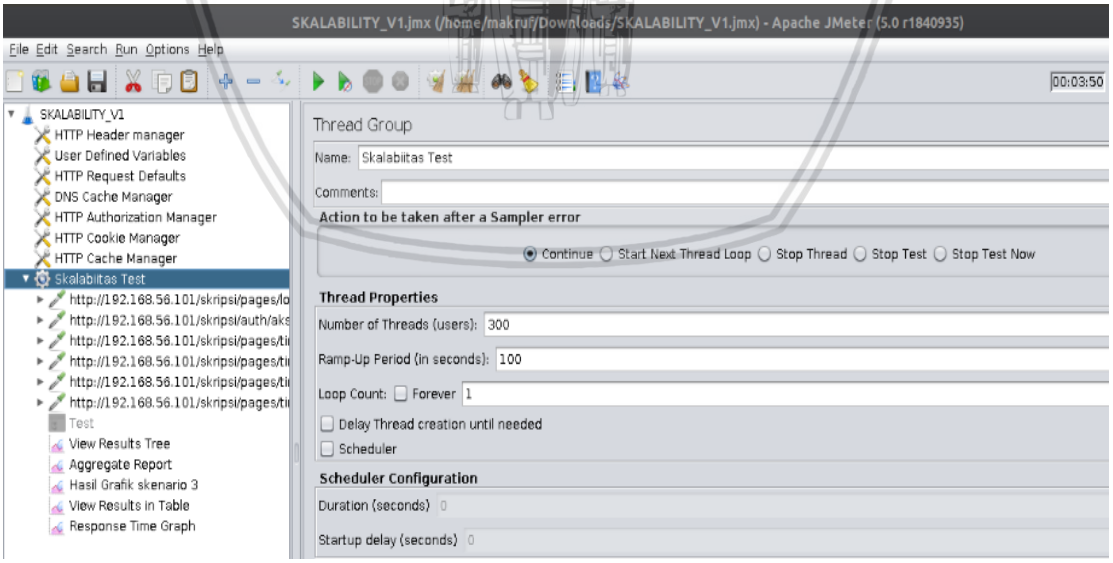
Bedasarkan hasil skenario yang ditampilkan melalui hasil gambar grafik dan hasil tabel dengan waktu tempuh dari empat kali percobaan dari masing-masing *server web* didapatkan waktu rata-rata selama 01 menit 57 detik dalam menyelesaikan setiap percobaan. Terjadi perubahan waktu respon dari 200 data *user* acak yang telah login berhasil namun peningkatan waktu respon dalam rentang waktu ± pada detik 60 hingga detik ke 110 dari total rata-rata waktu tanggap yang dicapai, perubahan tersebut disebabkan dua *server web* yang nonaktif dan permintaan *session* di *handle* oleh *server web* yang lain secara acak. Dengan hasil ini maka menunjukkan bahwa aplikasi uji berhasil menjalankan skenario yang kedua. Pada skenario ini saat menjalankan *share session* dengan aktif atau non aktifnya dari salah dua *server web* akan mempengaruhi waktu respon pada kinerja sebuah *website*. Untuk detail seluruh hasil tabel dan grafik yang berhasil dari aplikasi pada pengujian skenario satu dapat dilihat pada Lampiran A.1.

Tabel 6.5 Tabel rata-rata *response time* dari skenario ke-2

Percobaan	*Server Web* yang Aktif	*Request*	Rata rata *response time* (ms)	Rata rata *response time* (min)	Status *server web*
Ke-1	2	800	60050 *milisecond*	1 *minute*	*Server web* 1 dan 2 mati
Ke-2	2	800	75450 *milisecond*	1,25 *minute*	*Server web* 2 dan 3 mati
Ke-3	2	800	72500 *milisecond*	1,20 *minute*	*Server web* 3 dan 4 mati
Ke-4	2	800	82930 *milisecond*	1,38 *minute*	*Server web* 4 dan 1 mati

6.2.4 Skenario ke-3 sistem *Shared Session*

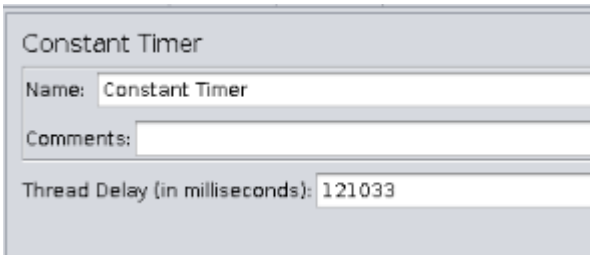
Pada skenario yang ketiga ini melakukan *request session* untuk dilihat dari seberapa handal dalam mendistribusikan *session* yang dibutuhkan. Skenario ini menggunakan 300 *user virtual* yang akan menjalankan rangkaian skenario dengan melakukan *request session* ke *database* dan diberikan *session ID* yang berbeda-beda agar mengetahui kinerja dari sistem *shared session*. Dalam skenario ini dikondisikan ada 3 *server web* yang dinonaktifkan dari 4 *server web* yang aktif. Pengujian dimulai dengan 300 *user* data melakukan *login* ke aplikasi *website* melalui *load balancing server* dengan IP 192.168.56.101 pada 8 Maret 2019 pukul 01:18. Dari 300 *user* akan secara acak *login* ke dalam aplikasi *website* yang akan dikelola *load balancing* yang berisikan 4 *server web* aktif dan disetiap 1 *user* melakukan *request* akan diberikan jeda waktu antar *user* lainnya dengan total waktu sebesar 100 detik.



Gambar 6.31 Tampilan konfigurasi *thread* pada skenario ke-3

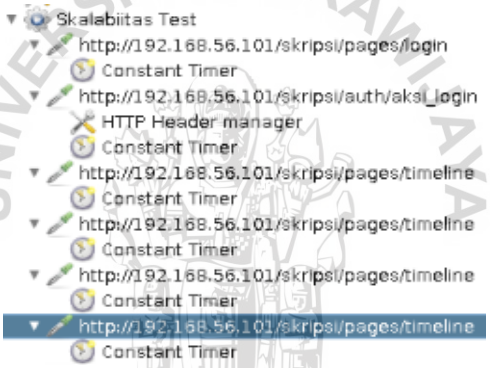
Konfigurasi request untuk menampilkan halaman *timeline* dan penentuan waktu *delay*. Waktu *delay* diberikan sebesar 121033 milidetik atau sebesar 121,033 detik

atau sebesar 2 menit. Waktu *delay* diberikan bertujuan untuk memberi jeda waktu sebelum *request* dieksekusi kemudian tiga dari empat *web server* tersebut akan



Gambar 6.32 Tampilan request halaman timeline dan waktu *delay*

Untuk pengaturan dan daftar konfigurasi *request* pada skenario ketiga tidak jauh berbeda dengan skenario yang pertama dan kedua. Perbedaan yang signifikan hanya pada jumlah *user* data dan *delay* waktu tiap *user*.

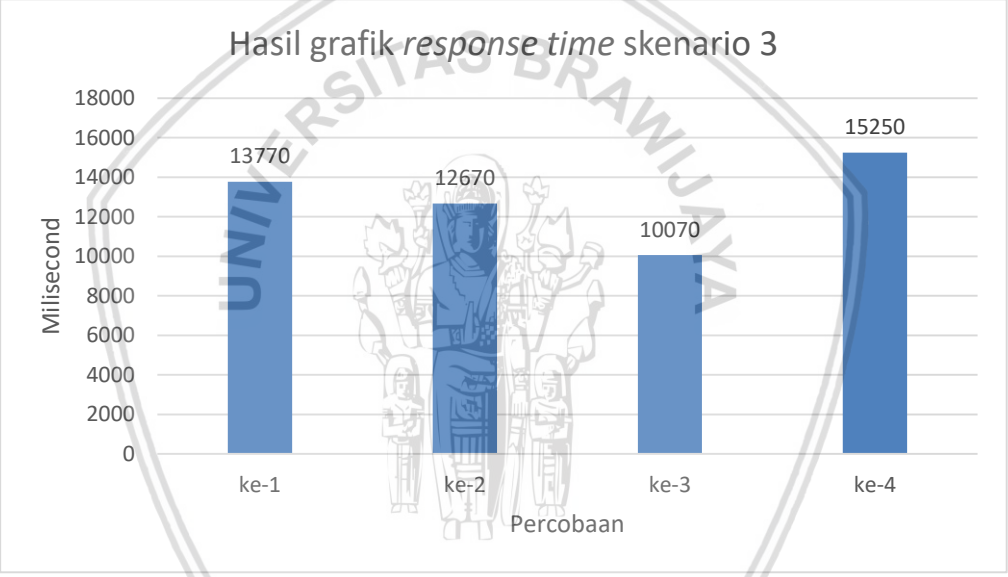


Gambar 6.33 Tampilan *HTTP request* pada skenario ke-3

Dalam skenario ini dilakukan empat kali percobaan untuk menguji kehandalan setiap masing-masing *server web* nya. Yang pertama dengan menonaktifkan layanan dari aplikasi *Apache server web* satu, dua dan tiga, kemudian jika sudah selesai pecobaan pertama dilanjutkan menonaktifkan aplikasi *Apache server web* kedua, ketiga dan empat lalu ini dilakukan hingga percobaan keempat. Saat skenario dalam proses eksekusi dan proses tersebut terjadi setelah 300 data user selesai melakukan proses *login* lalu sebelum masuk ke halaman *timeline*, peneliti menonaktifkan layanan aplikasi Apache dari *server web* secara manual. Berikut hasil data dari empat kali percobaan melalui aplikasi pengujian.

Tabel 6.6 Tabel hasil *response time* dari skenario ke-3

Percobaan	*Request*	Hasil *server web* dalam *response time* (ms)				Total Hasil Percobaan	Status *server web* bedasarkan percobaan	Keberhasilan
		1	2	3	4			
Ke-1	1200	0	0	0	13770	13770	*Server web* 1, 2 dan 3 mati	Tidak ada *erorr*
Ke-2	1200	12670	0	0	0	12670	*Server web* 2, 3 dan 4 mati	Tidak ada *erorr*
Ke-3	1200	0	10070	0	0	10070	*Server web* 3, 4 dan 1 mati	Tidak ada *erorr*
Ke-4	1200	0	0	15250	0	15250	*Server web* 4, 1 dan 2 mati	Tidak ada *erorr*



Gambar 6.34 Hasil grafik *response time* dari skenario ke-3

Bedasarkan hasil skenario yang ditampilkan melalui hasil gambar grafik dan hasil tabel dengan waktu tempuh sejumlah empat kali percobaan dari masing-masing *server web* didapatkan waktu rata-rata selama 03 menit 50 detik dalam menyelesaikan setiap percobaan. Terjadi perubahan waktu respon dari 300 data *user* acak yang telah *login* berhasil namun peningkatan waktu respon dalam rentang waktu ± pada detik 132 hingga detik ke 230 dari total rata-rata waktu tanggap yang dicapai, perubahan tersebut disebabkan tiga *server web* yang nonaktif dan permintaan *session* di *handle* oleh satu *server web* dalam satu klaster. Dengan hasil ini maka menunjukkan bahwa aplikasi uji berhasil menjalankan skenario yang ketiga. Pada skenario ini saat menjalankan *share session* dengan aktif atau non aktifnya dari salah tiga *server web* akan mempengaruhi waktu respon pada kinerja sebuah *website*. Untuk detail seluruh hasil tabel dan grafik yang berhasil dari aplikasi pada pengujian skenario satu dapat dilihat pada Lampiran A.1.

Tabel 6.7 Tabel rata-rata *response time* dari skenario ke-3

Percobaan	*Server Web* yang Aktif	*Request*	Rata rata *response time* (ms)	Rata rata *response time* (sec)	Status *server web*
Ke-1	1	1200	13770 milisecond	13,77 second	*Server web* 1, 2, 3 mati
Ke-2	1	1200	12670 milisecond	12,67 second	*Server web* 2, 3, 4 mati
Ke-3	1	1200	10070 milisecond	10,07 second	*Server web* 3, 4, 1 mati
Ke-4	1	1200	15250 milisecond	15,25 second	*Server web* 4, 1, 2 mati

BAB 7 PENUTUP

7.1 Kesimpulan

Bedasarkan implementasi sistem dan analisis hasil pengujian dari kinerja sistem *shared session* yang berperan sebagai sistem perantara database dengan seluruh *server web* aktif pada satu klaster *server web*, didapatkan beberapa kesimpulan yang diantaranya sebagai berikut:

1. Mekanisme *shared session* yang sudah diimplementasikan dalam penelitian ini dapat memastikan efisiensi dari penggunaan sebuah *website* bagi *user* yang telah melakukan *login* ke dalam aplikasi *website* melalui satu *server web* dengan tidak perlu melakukan *login* kembali meskipun mengakses halaman *website* dan *request 'session'* tersebut dikelola oleh *server web* yang berbeda di waktu yang sama. Proses tersebut akan bisa bekerja dengan baik karena didukung oleh mekanisme *Round Robin* yang aktif pada *load balancing* dan *database MySQL* yang dapat menyimpan data *session* serta menyediakan cadangan data *session* untuk dibagikan ke seluruh *server web* yang membutuhkan.
2. Dari hasil pengujian sistem *shared session* pada aplikasi *website* yang telah dilakukan sebelumnya, didapatkan beberapa hasil sebagai berikut:
   a. Pengujian *Login* dan *Logout* dari sistem *Shared Session*

      Terdapat dua skema pengujian yakni pengujian *login* dan pengujian *logout* dari sistem *shared session* ini sebagai perantara sistem yang mengatur dimana *session* dapat disimpan dan dibagikan melalui database. Bedasarkan hasil pengujian *login* dan pengujian *logout* yang dilakukan dan direncanakan semuanya berhasil dipenuhi melalui aplikasi *website*.

   b. Pengujian *black box* sistem *Shared Session*

      Bedasarkan hasil pengujian *black box* yang dilakukan melalui 3 skenario berbeda dan di masing-masing skenario yang telah dijalankan berhasil dipenuhi tanpa ada kesalahan pada aplikasi. Dari hasil pengujian terdapat variasi *response time* dalam sistem yang ditunjukkan pada aplikasi uji dapat disimpulkan bahwa tidak aktifnya dari salah satu sampai tiga hingga keempat *server web* sangat mempengaruhi waktu merespon dari *server web* yang telah meng*handle* dari *server web* yang nonaktif.

   c. Hasil pengujian *black box*

      Dari pengujian yang dilakukan melalui 3 skenario berbeda maka dapat diketahui skalabilitas sistem, yang mana hasil tersebut mendapatkan nilai *response time* yang beragam.

- Skenario ke 1 dengan 4 kali percobaan yang menguji setiap *node server web* disaat 1 *server web* nonaktif didapatkan waktu merespon sebesar 18,7 *second* sampai dengan 26,2 *second*, dengan waktu masing-masing percobaan selama 01 menit 20 detik berdasarkan 100 *user* yang melakukan *login* ke dalam *website*.
- Skenario ke 2 dengan 4 kali percobaan yang menguji setiap *node server web* disaat 2 *server web* nonaktif didapatkan waktu merespon sebesar 1 *minute* sampai dengan 1,38 *minute*, dengan waktu masing-masing percobaan selama 01 menit 57 detik berdasarkan 200 *user* yang melakukan *login* ke dalam website.
- Skenario ke 3 dengan 4 kali percobaan yang menguji setiap *node server web* disaat 3 *server web* nonaktif didapatkan waktu merespon sebesar 10,07 *second* sampai dengan 15,25 *second*, dengan waktu masing-masing percobaan selama 03 menit 50 detik berdasarkan 300 *user* yang melakukan *login* ke dalam *website*.

7.2 Saran

Saran dalam penelitian ini yang kelak kedepannya berguna untuk perbaikan dan pengembangan penelitian lebih lanjut.

Penggunaan metode pendukung penyimpanan data pada sistem *shared session* ini yakni menggunakan *Redis server*. Penerapan tersebut bertujuan untuk skala penggunaan dan penyimpanan data *session* yang lebih cepat dibandingkan menggunakan *database MySQL*. Karena penggunaan *database MySQL* lebih *persistent* dan membutuhkan ruang penyimpanan *memory* yang lebih besar di dalam *server*.

DAFTAR PUSTAKA

Ansharullah, K. (2016). Implementasi Sistem Load Balancing Dengan Algoritma Round Robin Untuk Mengatasi Beban Server di SMK Negeri 4 Kudus.

Ardhitya, A. I. (2007). Pengertian Mesin Virtual Pengertian Mesin Virtual.

Dwi Prastyo, A. (2017). Implementasi Moodle Menggunakan Load Balancing, Failover, Dan Shared Session. *Bandung, Universitas Telkom*, 302. Retrieved from https://openlibrary.telkomuniversity.ac.id/pustaka/139245/implementasi-moodle-menggunakan-load-balancing-failover-dan-shared-session.html

Erawan, L. (2014). Dasar-Dasar Php, 1–47.

Giurca, A. (2006). Forms, Cookies and Sessions, 1–21.

Griffiths, A. (2010). *Codeigniter 1.7 Professional Development*. Retrieved from https://www.ccr.ro/files/products/codeigniter-1.7-professional-development_.pdf

Julianto, R. (2017). Implementasi Load Balancing Di Web Server Menggunakan Metode Berbasis Sumber Daya CPU Pada Software Defined Networking. *Malang, Fakultas Ilmu Komputer*.

Kurniawan, A. A. (2012). *Penjadwalan Proses Dan Algoritma Penjadwalan Proses*. Semarang.

Lahtinen, E. (2014). USING SESSION REPLICATION IN WEB SERVICES, (December).

Lukitasari, D., & Oklilas, A. fali. (2013). Analisis Perbandingan Load Balancing Web Server Tunggal Dengan Web server Cluster Menggunakan Linux Virtual Server. *Generic*, *5*(2), 31–34. Retrieved from http://eprints.unsri.ac.id/336/1/6-.pdf

Mukhlis, A. (2018). SESSION DAN COOKIE. *Makassar, Politeknik Negeri Ujung Pandang*.

Nixon, R. (2014). *Learning PHP MySQL JavaScript With jQuery CSS HTML5 4th Edition*.

Reza Palevi, A. K. (2013). ANALISIS DAN PERANCANGAN SISTEM INFORMASI PENERIMAAN PESERTA DIDIK BARU BERBASIS WEBSITE PADA SMP NEGERI 2 MOJOSONGO Pendahuluan Landasan Teori & Tinjauan Umum. *Jurnal Ilmiah DASI*, *14*(04), 2–7.

Rosalia, M., Munadi, R., & Mayasari, R. (2016). Implementasi High Availability Server Menggunakan Metode Load Balancing dan Failover pada Virtual Web Server Cluster. *E-Proceeding of Engineering*, *3*(3), 4496–4503.

Safitri, L. (2012). Analisis Jaringan Komputer menggunakan Teknologi Virtualisasi, (24), 1–9.

Saputro, T. H. D. (2014). Pemodelan CPU Scheduling dengan Algoritma Round Robin sebagai Media Pembelajaran Mata Kuliah Sistem Operasi, 1–8.

Sasongko, A. (2014). Web-Bug Dengan Memanfaatkan Variable Server PHP Untuk Mengumpulkan Informasi Aktifitas Pengunjung Website. *Pontianak, AMIK "BSI Pontianak," 2*(1), 561–565.

Silitonga, J., Suswaini, E., & Kurniawan, H. (n.d.). A . Apache Web Server. *Teknik Perangkat Lunak, Fakultas Teknik, Universitas Maritim Raja Ali Haji Jl.*, 3–5.

Singh, H., & Kumar, S. (2015). WSQ: Web Server Queueing Algorithm for Dynamic Load Balancing. *Wireless Personal Communications*, *80*(1). https://doi.org/10.1007/s11277-014-2005-7

Widigdo, A. K. (2012). Dasar Pemrograman PHP dan MYSQL. *Dasar Pemrograman PHP Dan MySql*, 1–29.

Yoga Hartomo, H. (2015). IMPLEMENTASI WEB SERVER LOAD BALANCING PADA MESIN VIRTUAL MAKALAH. *Surakarta, UNIVERSITAS MUHAMMADIYAH*, *49*(23–6).